

EDIÇÃO DE ANIVERSÁRIO  
CASA DA ROBÓTICA



VAMOS JOGAR!



---

# CRIANDO JOGOS COM ARDUINO

PASSO A PASSO

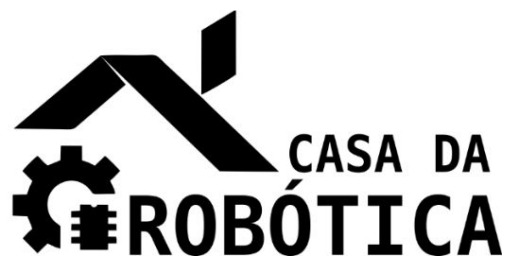
---

INICIAR JOGO









Copyright © 2020 da WL Componentes Eletrônicos Ltda.

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/1998. É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização, por escrito dos autores.

Editores: Luan Silva Santana e Welligton Assunção Azevedo

Direção de arte: Edvan da Silva Oliveira

Diagramação e capa: Carol Correia Viana

Produção e revisão: Carol Correia Viana, Kleber Rocha Bastos, Luan Silva Santana e Welligton Assunção Azevedo

Colaboração: Carlos Dyorgenes Santana

Primeira Edição

WL Componentes Eletrônicos

CNPJ: 29.495.665/0001-03

Avenida Brumado 1400

46052-000 - Vitória da Conquista, BA - Brasil

Tel.: +55 77 99151-2820

E-mail: [contato@casadarobotica.com](mailto:contato@casadarobotica.com)

Site: [www.casadarobotica.com](http://www.casadarobotica.com)

# APRESENTACAO

Olá,

Obrigado por adquirir nosso e-book: **Criando Jogos com Arduino – Passo a Passo.**

Este e-book comemorativo dos 5 anos da Casa da Robótica foi elaborado para você, que assim como nós, é apaixonado por eletrônica, programação e games.

Pensando nisso, compilamos três jogos que podem ser desenvolvidos utilizando a placa microcontroladora UNO SMD R3 Atmega328, compatível com o projeto Arduino UNO, ou com qualquer outra placa Arduino.

Esperamos que você aproveite este material com entusiasmo e que ele auxilie a sua jornada de estudos.

Um grande abraço

## **Equipe Casa da Robótica**

# SUMÁRIO

1. [INTRODUÇÃO](#)
2. [PLATAFORMA ARDUINO](#)
  - [O QUE É O ARDUINO?](#)
  - [EXPLORANDO UMA PLACA UNO SMD R3 ATMEGA328](#)
  - [PRIMEIROS PASSOS](#)
  - [EXPLORANDO O ARDUINO IDE](#)
3. [FUNDAMENTOS DE PROGRAMAÇÃO](#)
  - [Estruturas básica de um Sketch](#)
  - [Variáveis](#)
  - [Funções](#)
  - [Bibliotecas](#)
  - [Operadores](#)
  - [Estrutura de controle de fluxo](#)
4. [CRIANDO JOGOS COM ARDUINO](#)
  - [STOP – UM JOGO COM LEDS](#)
  - [JOGO DA MEMÓRIA](#)
  - [BATALHA ESPACIAL](#)
  - [DESAFIOS](#)
5. [CONSIDERAÇÕES FINAIS](#)





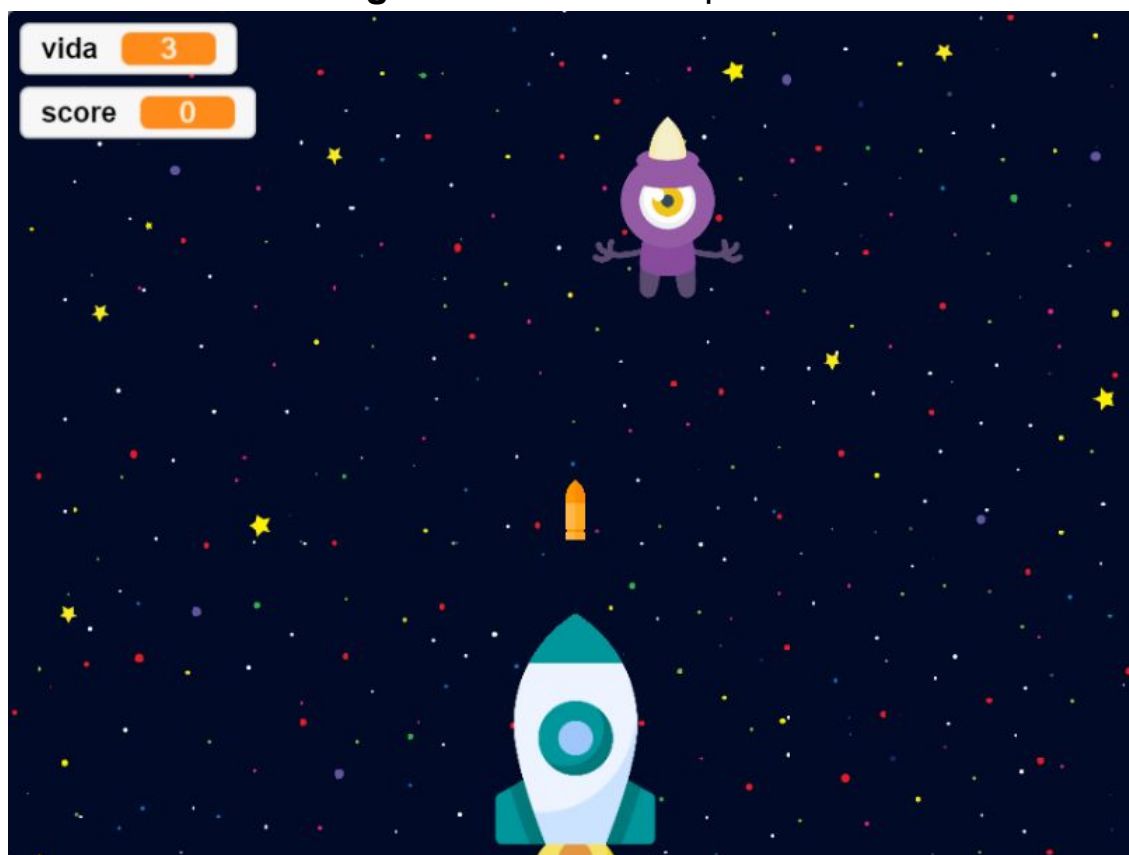


# 1. INTRODUÇÃO

Já parou para pensar que criar seu próprio jogo pode ser bastante divertido? Com o uso de placas do projeto Arduino é possível desenvolver uma infinidade de jogos.

A proposta desse e-book é apresentar a você três jogos que foram desenvolvidos utilizando a plataforma Arduino e outros componentes básicos de eletrônica. Estes jogos serão: Batalha Espacial (**Figura 1**), Stop – Um jogo com LEDs e Jogo da Memória.

**Figura 1** - Batalha Espacial.



Mas que componentes vamos usar? A Tabela 1 exibe a lista de materiais necessários para construção dos jogos:

**Tabela 1** - Lista de Componentes.

| COMPONENTES                                                                     |
|---------------------------------------------------------------------------------|
| <a href="#">Placa UNO SMD R3 Atmega328 compatível com o projeto Arduino UNO</a> |
| <a href="#">Cabo USB (tipo A para tipo B)</a>                                   |
| <a href="#">Cabos jumpers macho-macho</a>                                       |
| <a href="#">Protoboard</a>                                                      |
| <a href="#">LEDs difuso de 5 mm</a>                                             |
| <a href="#">Resistores de 220 <math>\Omega</math></a>                           |
| <a href="#">Resistores de 10 k<math>\Omega</math></a>                           |
| <a href="#">Botões do tipo Push Button</a>                                      |
| <a href="#">Buzzer ativo</a>                                                    |

Nesse e-book você também encontrará alguns conceitos importantes para iniciar seus estudos em eletrônica e programação. Para isto, inicialmente, expomos um pouco sobre a plataforma Arduino e o microcontrolador UNO R3 Atmega328, compatível ao projeto Arduino UNO.

O Capítulo 3 foi preparado para auxiliar você a aprender os principais fundamentos da linguagem de programação utilizada pelo Arduino, como declaração e tipos de variáveis, estrutura de condição e repetição, funções e biblioteca.

O Capítulo 4 tem como objetivo reunir os conhecimentos adquiridos e nos divertir com a criação dos jogos Stop – Um jogo com

LEDs, Jogo da Memória e Batalha Espacial.





# PLATAFORMA ARDUINO



# O QUE É O ARDUINO?

O Arduino é uma plataforma eletrônica de código aberto baseada em software e hardware de fácil utilização, sendo ideal para iniciantes e para qualquer pessoa que deseja construir projetos eletrônicos.

As placas Arduino permitem a conexão de circuitos eletrônicos aos seus terminais, o que possibilita a leitura de entradas – luminosidade em um sensor, o acionamento de um botão ou uma mensagem SMS, e transformar estas informações em uma saída controlando algum dispositivo – por exemplo ligando um LED, ativando um motor ou enviando uma mensagem.

As placas Arduino podem ser conectadas ao computador por meio do barramento serial universal (USB), possibilitando sua utilização como placa de interface e controlar dispositivos por meio do seu computador.

A plataforma Arduino oferece uma série de vantagens em relação a outras plataformas, o que o tornou popular entre professores, alunos, amadores e projetistas, tais como:

- Possuir ambiente multiplataforma, ou seja, pode ser executado nos principais sistemas operacionais comercializáveis;
- Contar uma IDE de programação própria;
- Poder ser programado utilizando um cabo USB;
- Possuir hardware e software de fonte aberta;
- Ter sido desenvolvido em um ambiente educacional, sendo ideal para iniciantes.

Diante da sua popularização, a plataforma Arduino cresceu e atualmente conta com diversas versões de mercado. A Figura 2 ilustra

algumas versões da placa Arduino.

**Figura 2** - Algumas versões da placa Arduino



Além disso, existem uma série de placas compatíveis com o projeto Arduino, uma vez que seu hardware é aberto a replica destas placas são permitidas e possuem as mesmas características, pinagens e forma de uso. A Placa UNO SMD R3 Atmega328, por exemplo, é compatível ao projeto do Arduino UNO.

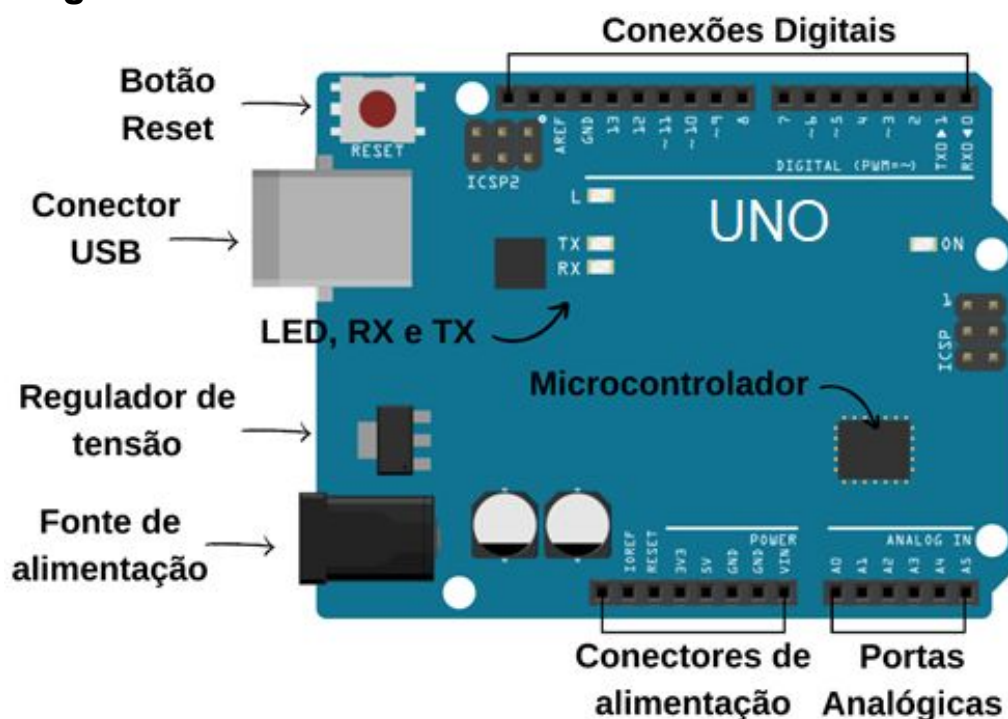
Existem placas Arduino bem pequenas (Nano, micro, mini), de tamanho médio e tradicional (Uno, Duemilanove, Leonardo), e as placas de maiores dimensões (Mega, Due). Diante de tanta variedade, você deve estar se perguntando: Qual placa devo usar no meu projeto? A escolha da versão ideal vai depender das necessidades de seu projeto, mas recomendamos:

- Placas de Arduino pequenas para projetos que precisam ser leves e ocupar pouco espaço;
- Placas de Arduino tamanho médio e tradicional para projetos de tamanho padrão como robôs, interfaces homem-máquina, central de monitoramento, entre outros;
- Placas de Arduino maiores dimensões para projetos que demandem de maior memória e número de portas de entrada e saída.

# EXPLORANDO UMA PLACA UNO SMD R3 ATMEGA328

Conhecer os elementos que compõe a placa UNO SMD R3 ATMEGA328 é de suma importância antes de iniciar os nossos projetos. Desta forma, vamos explorar esta placa microcontroladora (Figura 3) para nos familiarizar com seus vários componentes.

**Figura 3** - Placa microcontroladora UNO SMD R3 ATMEGA238.



Fonte de alimentação

O circuito interno da placa UNO deve ser alimentado com uma tensão contínua de 5V. Você pode alimentá-lo conectando-o a uma porta USB do computador, que fornecerá a alimentação e também a

comunicação de dados, ou por meio de uma fonte de alimentação externa, que forneça uma saída contínua entre 7 V e 12 V, por meio da utilização de um plug P4 ou o pino Vin.

## Regulador de tensão

Na placa UNO, o regulador de tensão tem como finalidade transformar qualquer tensão (entre 7 V e 12 V) que esteja sendo fornecida pelo conector de alimentação externa em uma tensão contínua de 5V.

## Conectores de alimentação elétrica

Os conectores de alimentação elétrica fornecem energia para dispositivos externos e são constituídos pelos pinos:

- Reset que possui a mesma função do botão Reset;
- 3,3 V e 5 V que fornecem tensão de 3,3 e 5 V, respectivamente;
- GND fornece potencial de terra aos dispositivos externos;
- Vin fornece ao dispositivo externo a mesma tensão que está sendo recebida pelo pino de alimentação externa.

## Entradas analógicas

A placa UNO possui 6 portas analógicas que estão indicados como Analog In, de A0 a A5. Esses pinos são dedicados a receber valores de grandezas analógicas, por exemplo, a tensão de um

sensor. As grandezas analógicas variam continuamente no tempo dentro de uma faixa de valores.

## Conexões digitais

A placa UNO possui 14 portas digitais que estão indicados como Digital, de 0 a 13. Estas portas podem ser utilizadas como receber ou enviar dados de grandezas digitais. Ao contrário das grandezas analógicas, as grandezas digitais não variam continuamente no tempo, mas sim entre valores definidos (0 ou 1, ligado ou desligado, sim ou não, 0 V ou 5 V).

Estes pinos digitais operam em 5V e corrente máxima de 40 mA. Além disso, alguns deles possuem funções especiais, como:

- Pinos 3, 5, 6, 9, 10 e 11 podem ser usados como saídas PWM, simulando uma saída analógica;
- Pinos 0 e 1 (RX e TX) podem ser utilizados para comunicação serial;
- Pinos 2 e 3 podem ser configurados para interrupção externa.

## Microcontrolador

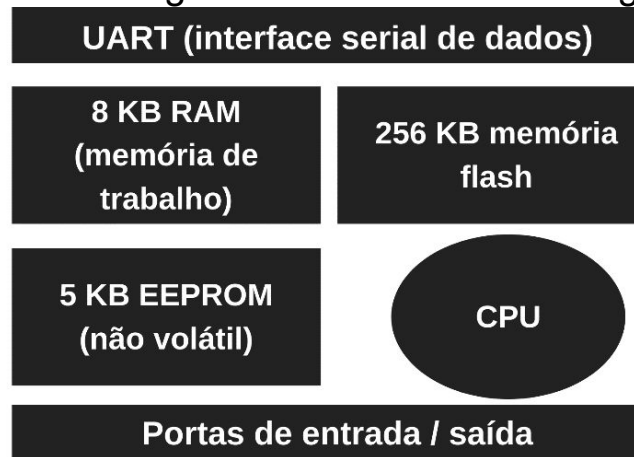
O microcontrolador utilizado na placa UNO é o ATmega328, um pequeno chip de 28 pinos que se encontra no centro da placa e é considerado o cérebro desse dispositivo. Esse único chip é um pequeno computador, contendo memória, processador e toda eletrônica necessária aos pinos de entrada e saída.

É no microcontrolador que tudo acontece, é nele que fica gravado o código desenvolvido para execução. O microcontrolador

permite que a placa Uno funcione de forma autônoma, em outras palavras, uma vez transferido o código não existe mais a necessidade de comunicação com o computador. Um fato que deve ser lembrado é que ao gravar um código, o anterior é descartado, ficando apenas o último código gravado.

Algumas características do microcontrolador ATmega328 encontram-se detalhadas na Figura 4.

**Figura 4** - Diagrama de blocos do ATmega328.



**Fonte:** Adaptado de Monk (2017).

Botão Reset

O botão Reset tem como única função reinicializar a placa microcontroladora.

Outros componentes

Além dos componentes citados, a placa UNO também conta com: um oscilador a cristal, capaz de realizar 16 milhões de ciclos ou oscilações por segundo; um conector serial de programação, outro meio de programar a placa UNO; e um chip de interface USB, que

converte os níveis de sinal usados pelo padrão USB em níveis que podem ser usados pela placa UNO.



# PRIMEIROS PASSOS

Para que a placa UNO execute qualquer ação você precisará escrever um código ou Sketch em linguagem C/C++ utilizando gratuitamente o software Arduino IDE, que se encontra disponível na versão online e offline.

O Arduino Web Editor é a interface de desenvolvimento online do Arduino, com ele é possível codificar, salvar os esboços na nuvem, fazer backup e enviar o código feito para qualquer placa compatível com o Arduino a partir do navegador de internet. Por estar hospedado online, o Arduino Web Editor estará sempre atualizado com os recursos, bibliotecas e suporte mais recente. Além disto, esta interface de desenvolvimento permite que você acesse um código salvo a partir de qualquer dispositivo conectado à internet. O Arduino Web Editor encontra-se disponível no link <https://create.arduino.cc/editor>.

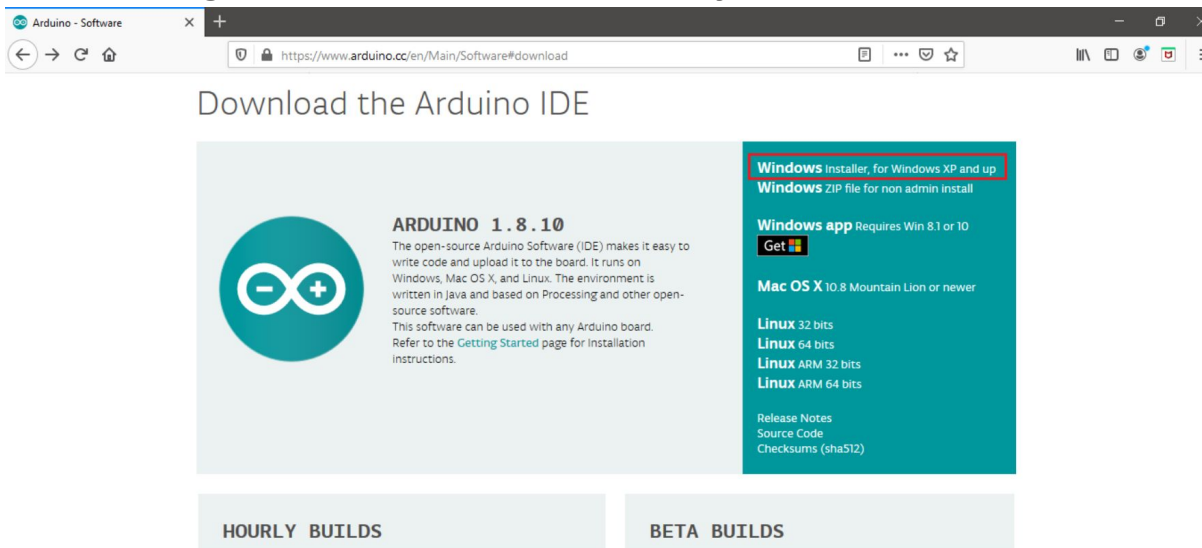
O Arduino IDE é a versão offline desta ferramenta de desenvolvimento e pode ser executado no Windows, Mac OS X e Linux. O download do Arduino IDE encontra-se disponível no link <https://www.arduino.cc/en/Main/Software#download>, em que faz-se necessária a escolha da versão apropriada para seu sistema operacional.

## Download e Instalação do Arduino IDE no Windows

A seguir, você encontrará o passo a passo para instalar o Arduino IDE no computador Windows.

1 - Acesse o link <https://www.arduino.cc/en/Main/Software#download> e escolha a opção “Windows Installer, for Windows XP and up”, conforme ilustra a Figura 5.

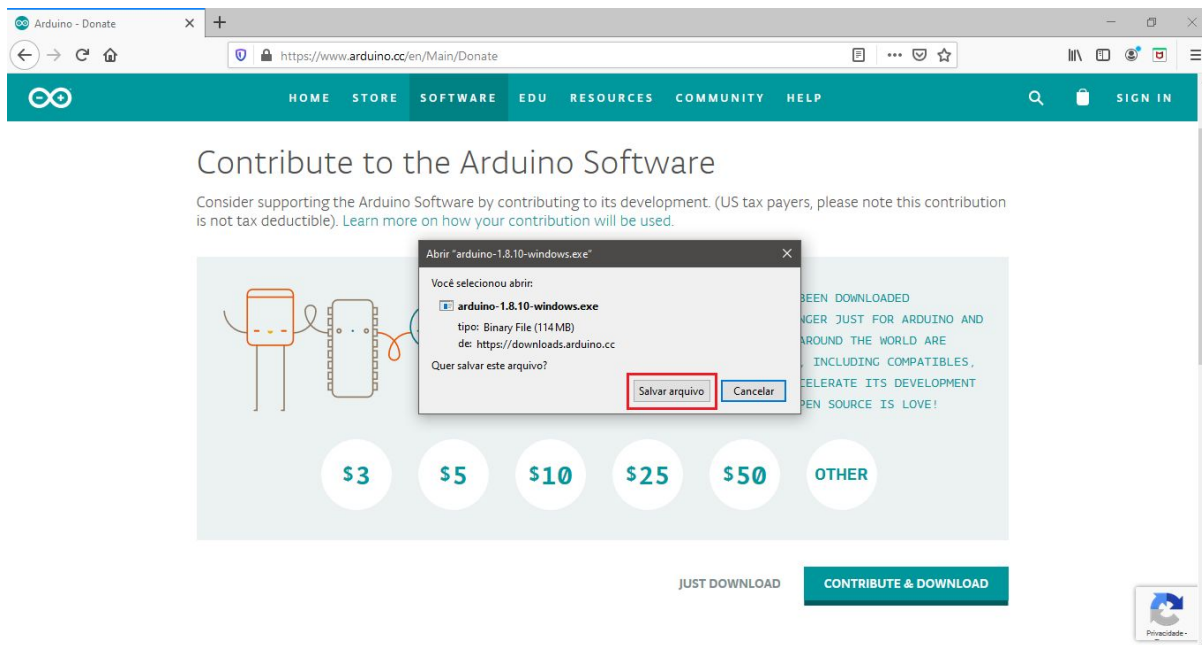
**Figura 5** – Passo 1 para instalação do Arduino IDE.



2 – Em seguida, para baixar gratuitamente o software Arduino IDE selecione a opção de Download.

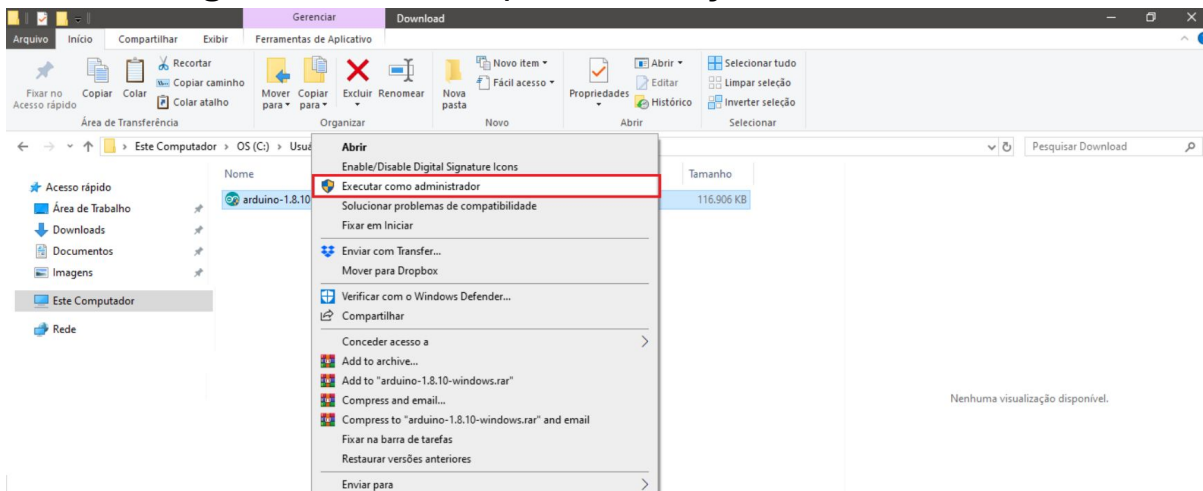
3 – Salve o arquivo do download e aguarde.

**Figura 6** - Passo 3 para instalação do Arduino IDE.



4 – Após conclusão do download, clique com o botão direito sobre o arquivo baixado e o execute como administrador, conforme Figura 7.

**Figura 7 - Passo 4 para instalação do Arduino IDE.**

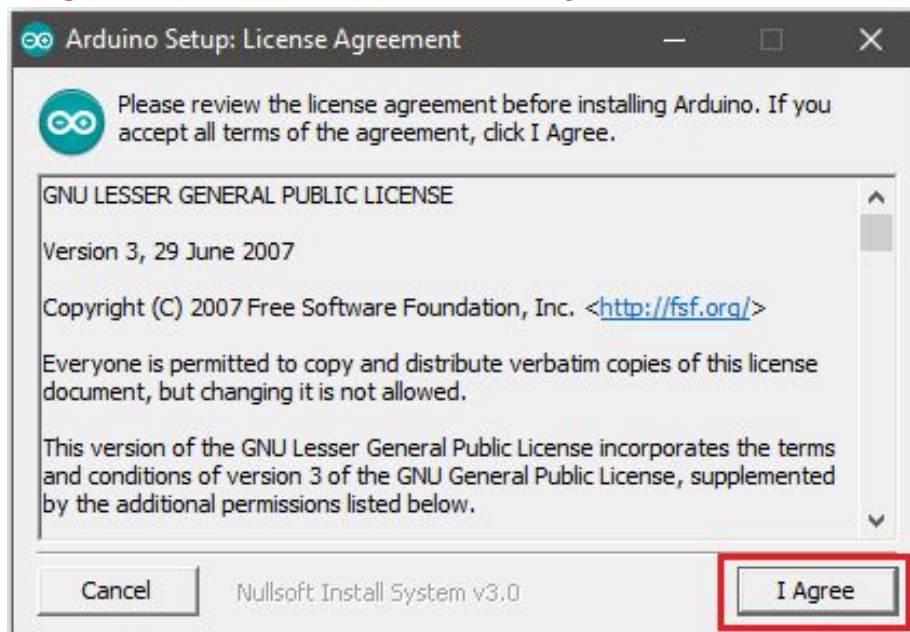


5 – Após isto, aparecerá uma tela do Controle de Conta do Usuário solicitando permissão para instalação do Arduino IDE com a seguinte

mensagem “*Deseja permitir que este aplicativo faça alterações no seu dispositivo?*”. Clique em SIM para iniciar a instalação.

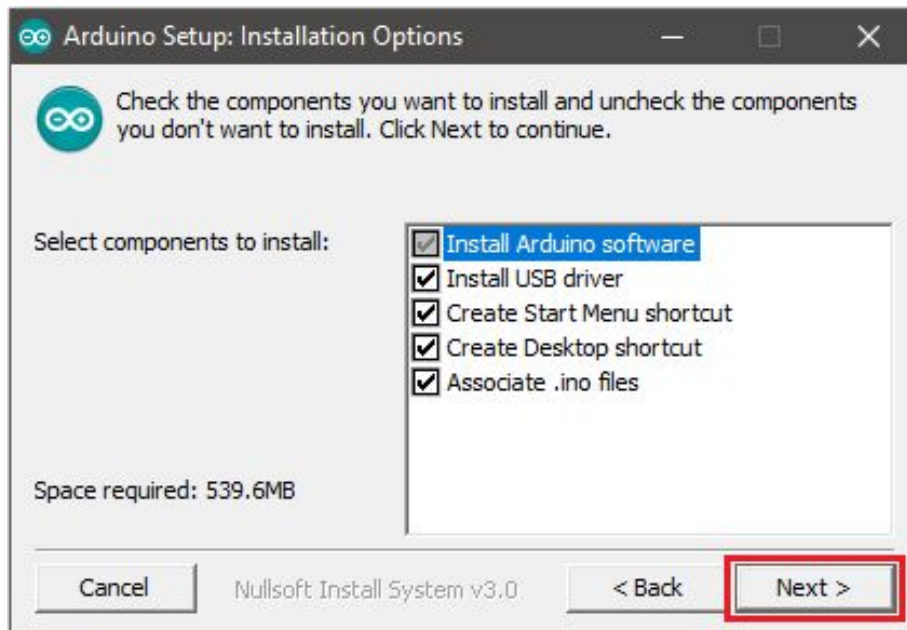
6 – A partir de então a tela de instalação do Arduino IDE será iniciada. A primeira ação que deve ser realizada para instalação deste software é aceitar os termos de licença clicando no botão “I Agree”, mostrado na Figura 8.

**Figura 8** - Passo 6 para instalação do Arduino IDE.



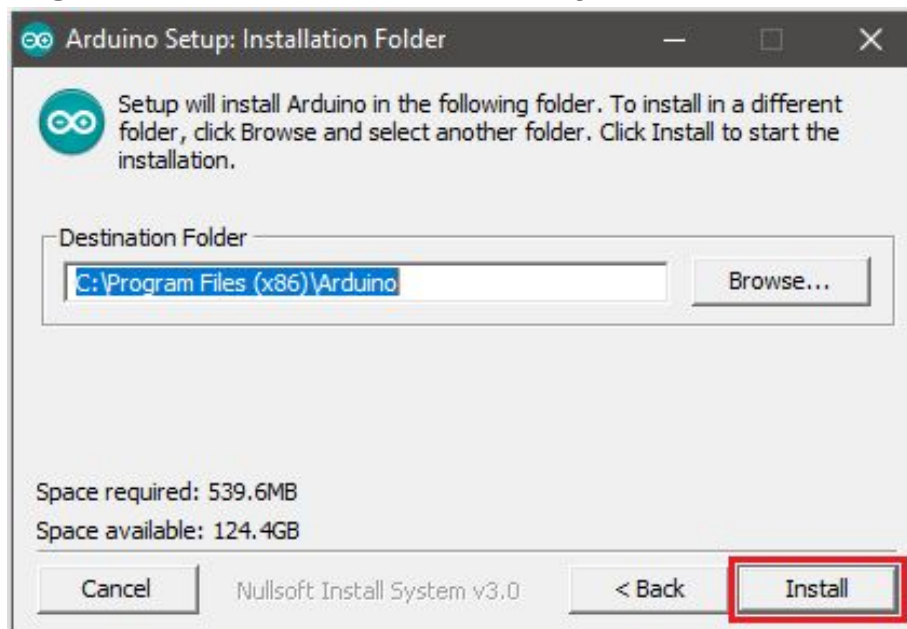
7 – Em seguida, você deve verificar se todos os itens estão selecionados para instalação e clicar no botão “Next >”, conforme a Figura 9.

**Figura 9** - Passo 7 para instalação do Arduino IDE.



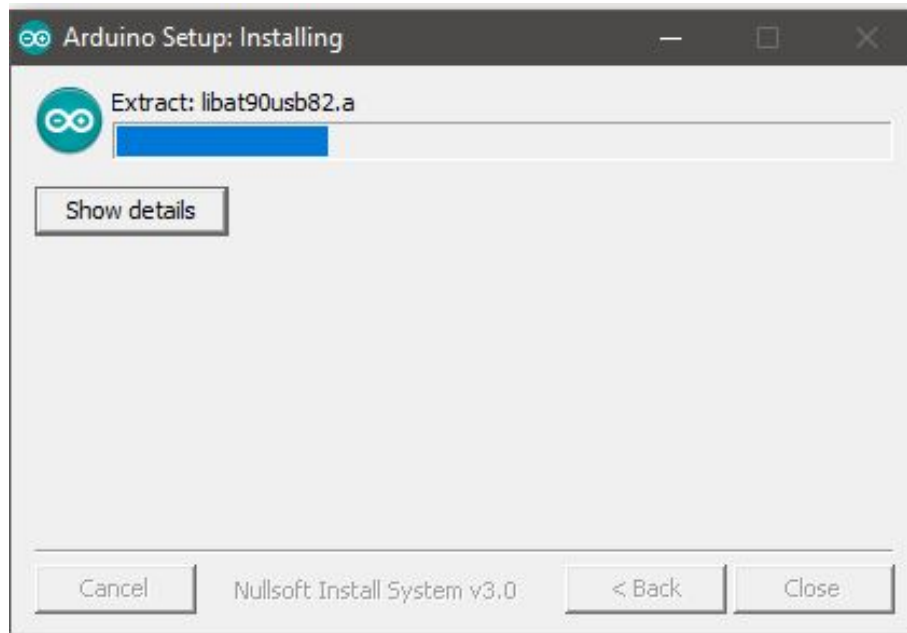
8 – Logo após, selecione o opção “Install” para proceguir com a intalação.

**Figura 10** - Passo 8 para instalação do Arduino IDE.



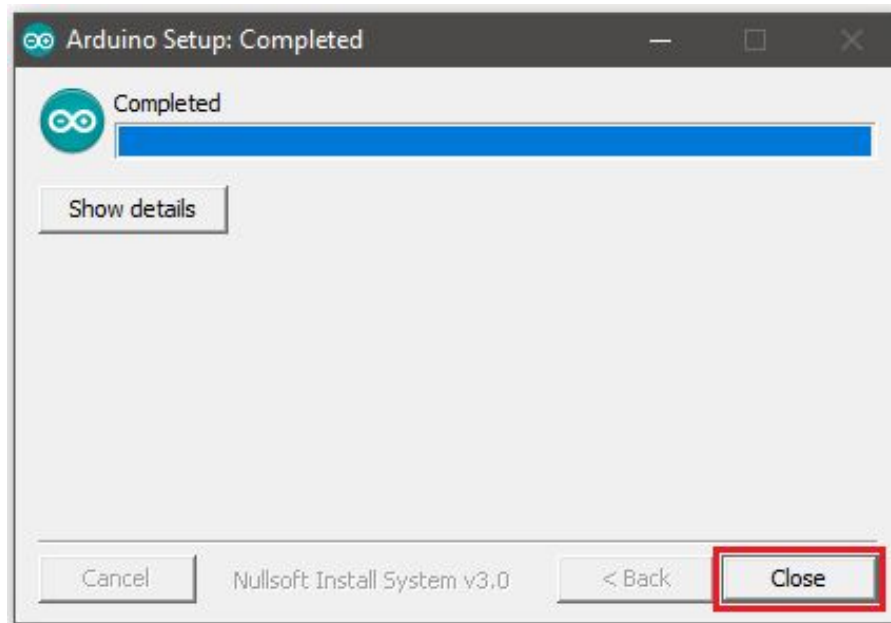
9 – Aguarde a conclusão da instalação. Este processo poderá ser acompanhado, conforme mostra a Figura 11.

**Figura 11** - Passo 9 para instalação do Arduino IDE.



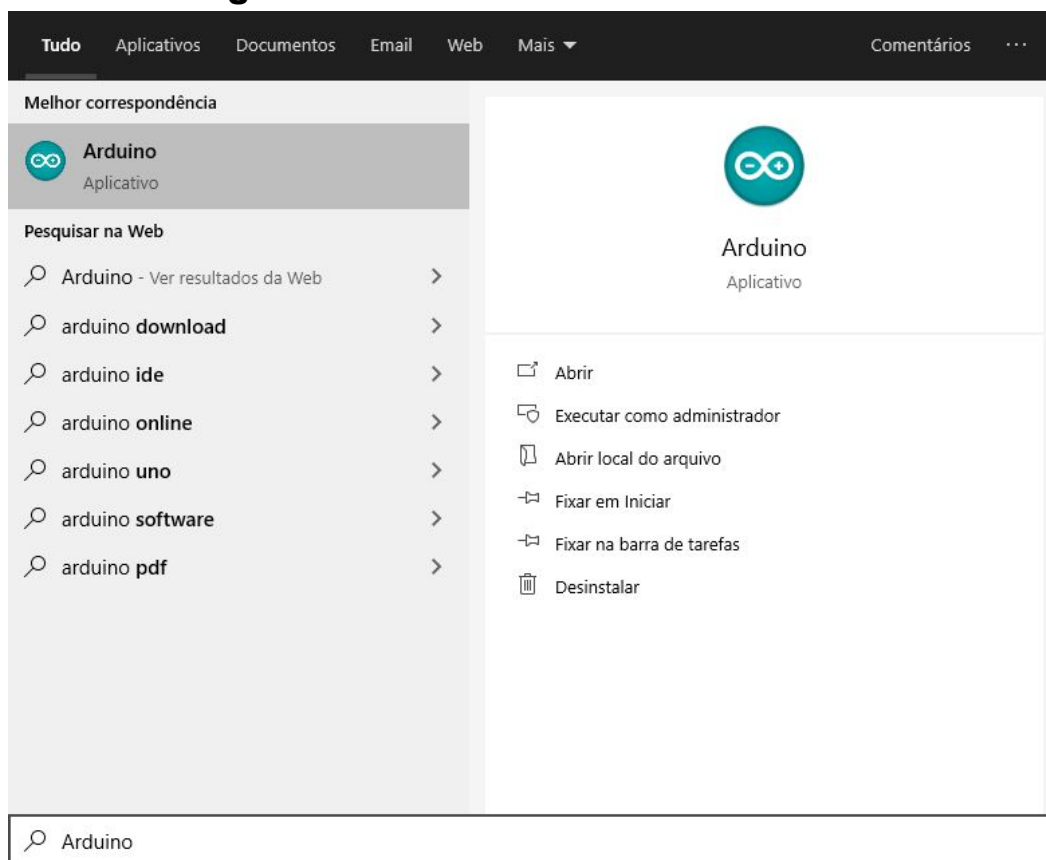
10 – Quando a instalação estiver completada, clique no botão “Close”.

**Figura 12** - Passo 10 para instalação do Arduino IDE.



Ocorrendo tudo bem na instalação do Arduino IDE, você pode inicializá-lo através do atalho criado na área de trabalho ou buscando por Arduino no menu iniciar, conforme Figura 13.

**Figura 13 - Inicializando o Arduino IDE.**



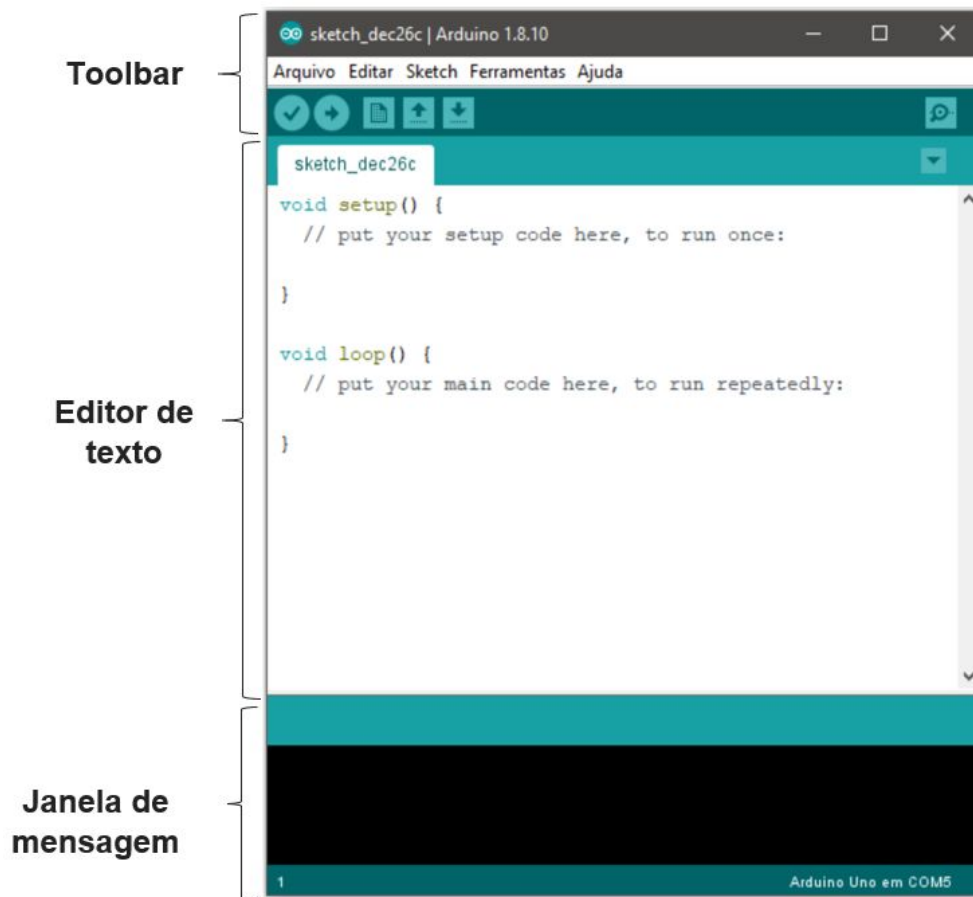




# EXPLORANDO O ARDUINO IDE

Ao abrir o Arduino IDE você verá uma tela semelhante à Figura 14. Caso esteja utilizando o Linux ou Mac OS X, podem haver pequenas diferenças, mas o IDE é basicamente o mesmo para todos os sistemas operacionais.

**Figura 14 - Visual do Arduino IDE.**



O Arduino IDE pode ser dividido em três partes: A Toolbar no início da tela, o editor de texto no centro e a janela de mensagens na base.

No top da Toolbar há uma barra de menus contendo comandos comuns com os itens: Arquivo, Editar, Sketch, Ferramentas e Ajuda. Os comandos e funções disponíveis na barra de ferramenta podem ser consultados ao acessar o comando Ajuda > Ambiente.

Ainda na Toolbar encontra-se os botões de atalho, que fornecem acesso rápido às funções mais utilizadas. A seguir são mostrados os ícones e o detalhamento de suas funções.



**Verificar:** Analisa se há erros em seu código



**Upload:** Compila seu código e o envia para a placa microcontroladora



**Novo:** Cria um novo Sketch



**Abrir:** Mostra uma lista de Sketch existentes



**Salvar:** Salva o seu Sketch atual



**Monitor serial:** Exibe os dados seriais enviados pela placa microcontroladora

O editor de texto é o campo destinado a escrita dos códigos. Os códigos escritos usando Arduino ou placas compatíveis são conhecidos como Sketches e são salvos com a extensão de arquivo .ino. Este editor de texto tem características de um editor tradicional, contendo funções de cortar, copiar, colar, selecionar tudo, entre outras.

A janela de mensagem fornece mensagens de feedback ao salvar e exportar arquivos, bem como exibe informações de erros no código ou ao compilar.

## Conectando a placa UNO ao computador

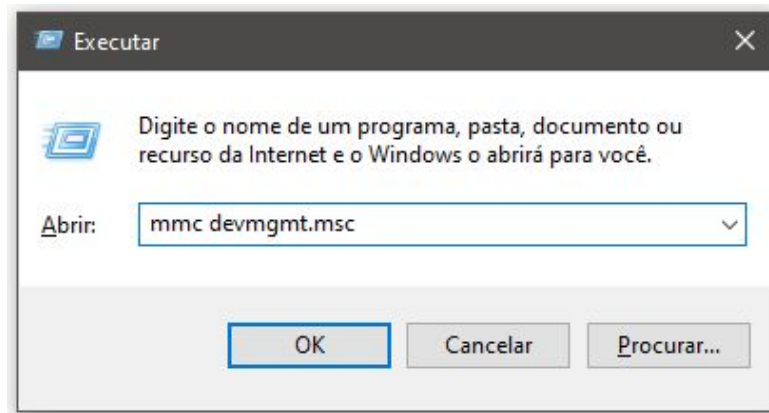
Agora que já conhecemos a placa UNO e sua interface de desenvolvimento, vamos conectá-lo ao computador. Esta conexão é realizada por meio de um cabo USB do tipo A para o tipo B, igual ao da Figura 15.

**Figura 15** - Cabo USB do tipo A para o tipo B.



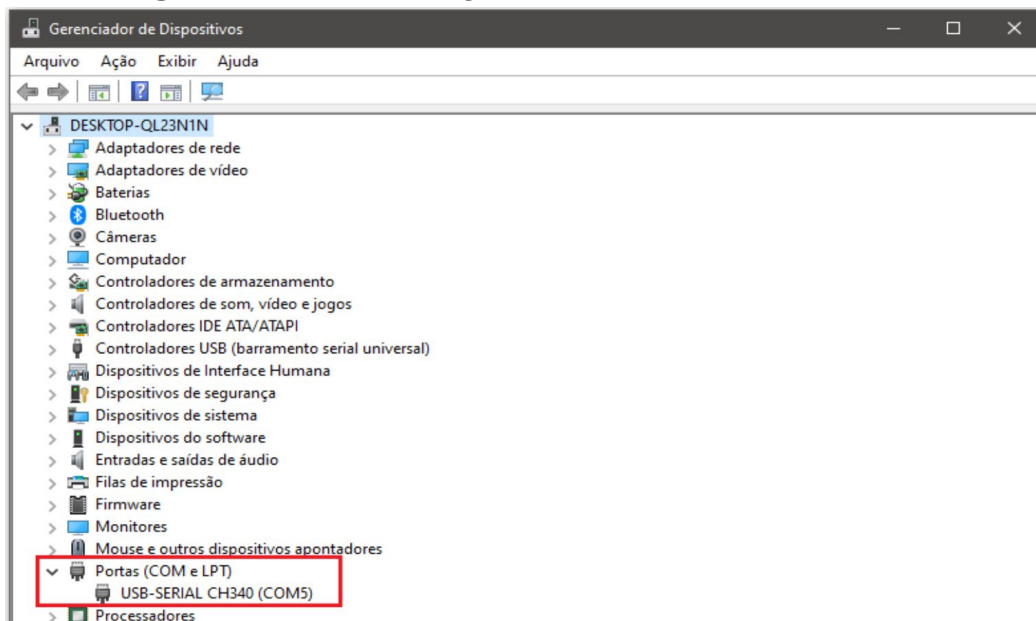
Conecte sua placa UNO ao seu computador através da USB. Para saber se o seu computador Windows está identificando a placa vamos realizar um teste acessando o gerenciador de dispositivos. Uma opção para se chegar neste painel é pressionar as teclas “Windows + r”. Assim que o menu executar abrir digite “mmc devmgmt.msc” sem as aspas, como se pode ser observado na Figura 16.

**Figura 16** - Atalho para acessar o gerenciador de dispositivos.



Após digitar esse comando e clicar em “OK” será aberta a tela da Figura 17. Para verificar se o driver da placa UNO foi reconhecido navegue até a opção Portas (COM e LPT) e expanda clicando na setinha do lado do nome. No exemplo abaixo a placa UNO foi reconhecida com sucesso pela porta COM de número 5, essa informação será útil posteriormente.

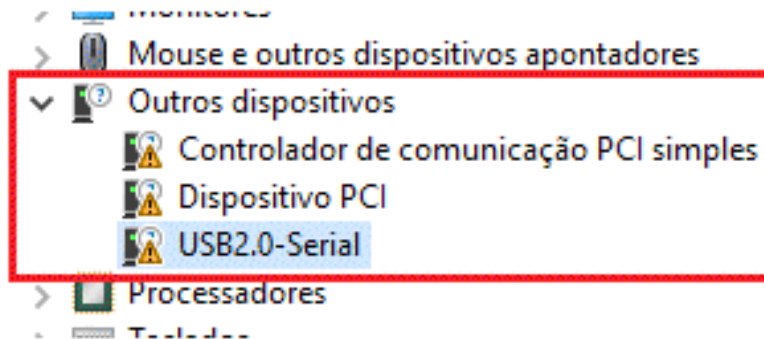
**Figura 17 - Tela do gerenciador de dispositivos.**



⚠️ Caso a placa UNO não seja reconhecida pelo seu computador, ela pode aparecer, com o ícone , em “Outros

dispositivos”, como na Figura 18.

**Figura 18** - Indicador de que a placa Arduino não foi reconhecida.

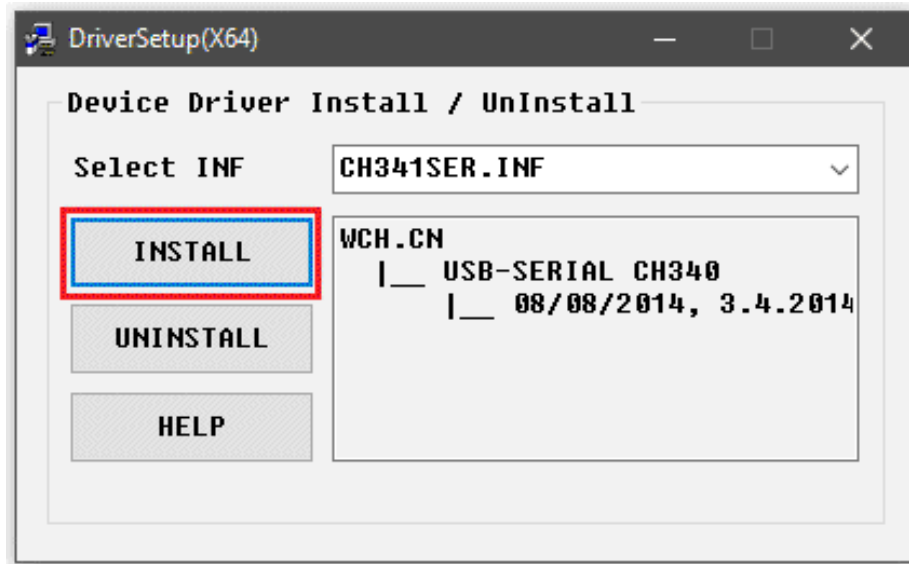


Isso acontece devido à falta de um driver para a interpretação do dispositivo. Para resolver esse problema basta baixar o driver disponível no link a seguir e executá-lo:

[www.blogdarobotica.com/download-driver-ch340](http://www.blogdarobotica.com/download-driver-ch340)

Após a conclusão do download, execute o arquivo baixado, instale o driver e aguarde a mensagem de confirmação da instalação, conforme Figura 19.

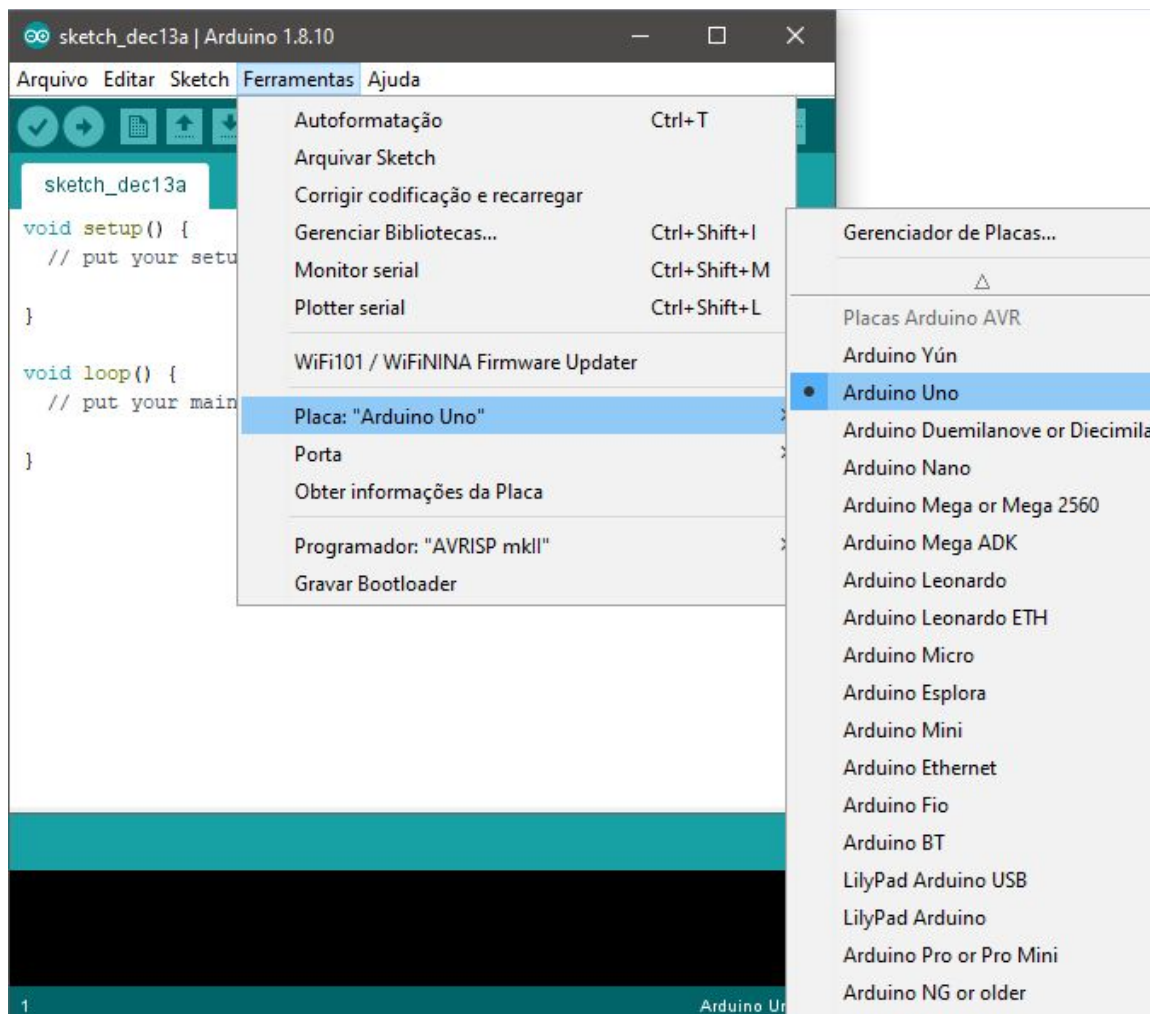
**Figura 19** - Instalação do driver para interpretação da placa UNO.



Seleção da placa e da porta de comunicação da placa UNO

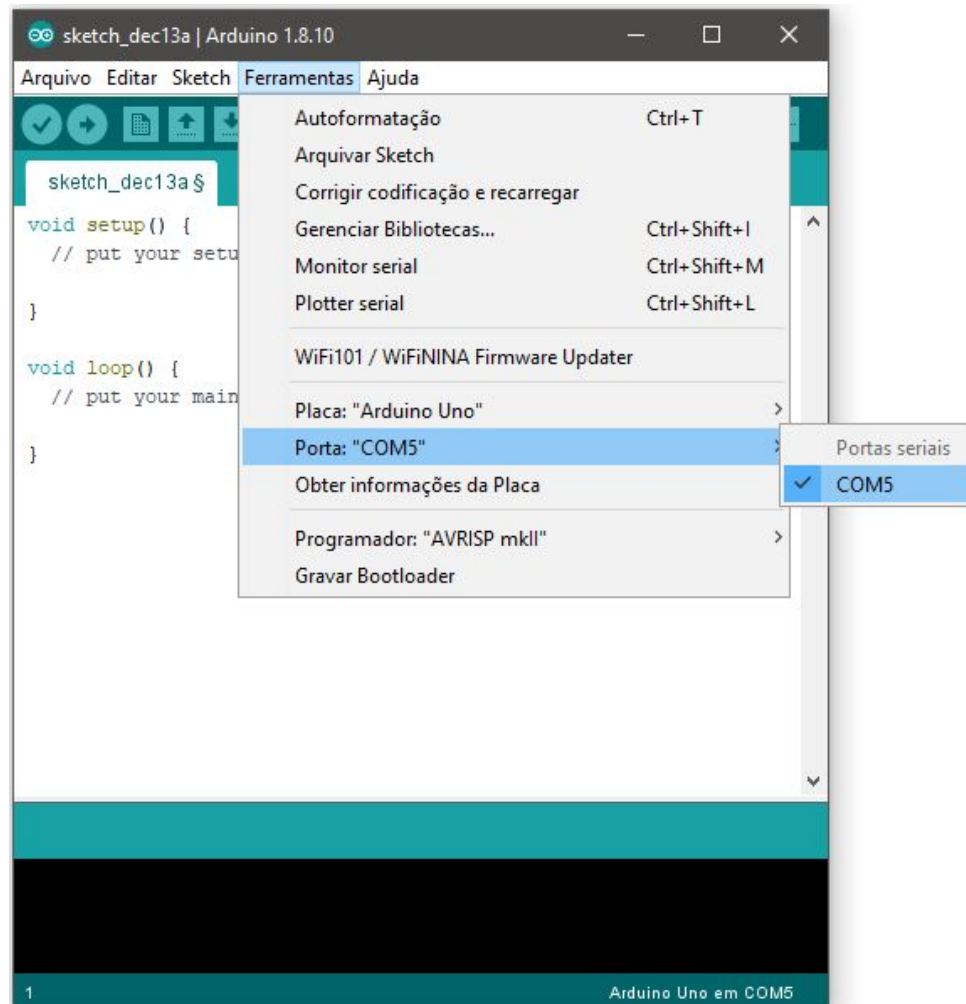
Agora que já temos a placa UNO conectada ao computador, vamos selecionar a placa e a porta de comunicação no Arduino IDE. Para tal, deve-se selecionar o modelo da placa utilizada no menu Ferramentas, para nossos exemplos usaremos o Arduino Uno, conforme Figura 20.

**Figura 20** - Seleção da placa Arduino.



Após a seleção do modelo, deve-se selecionar a porta de comunicação a placa foi atribuída, ou seja, a porta que a placa UNO foi reconhecida. Como vimos anteriormente, em nosso exemplo, a placa UNO foi reconhecida pela COM de número 5. A Figura 21 mostra a seleção da COM através do menu Ferramentas.

**Figura 21** - Seleção da porta COM.



Após isto, sua placa UNO estará pronta e o ambiente de desenvolvimento configurado para uso.







### 3. FUNDAMENTOS DE PROGRAMAÇÃO

Programação pode ser definida como o processo de projetar, escrever, verificar e manter um código ou programa que comande as ações de uma máquina, como computador, celular, microcontrolador, entre outras.

Um código, programa ou sketch – como são denominados os códigos em Arduino, são compostos por um conjunto de instruções sequenciais definido pelo programador que descrevem as tarefas que a máquina deve realizar. Em outras palavras, para que a máquina execute comandos específicos é necessário elaborar um programa escrito contendo todas as instruções, ordenadas sequencialmente, em uma linguagem de programação.

Uma linguagem de programação é um idioma artificial desenvolvido para prover a comunicação entre uma pessoa (programador) e uma máquina (computador, microcontrolador, etc.). Atualmente, existem centenas de linguagens de programação e em todas elas o programador pode definir como a máquina deve atuar, como armazenar ou transmitir os dados, quais ações tomar em diferentes situações, quando finalizar a sua operação, entre outras. Dentre as linguagens de programação mais populares pode-se citar o C, C++, C#, Java, JavaScript, Python, PHP.

Os programas para microcontroladores compatíveis com o projeto Arduino, como o microcontrolador UNO R3 Atmega328, são implementados tendo como referência a linguagem de programação

C++, mantendo preservada sua sintaxe clássica de declaração de variáveis, nos operadores, nas estruturas e em muitas outras características.

Antes de aprendermos a linguagem de programação do Arduino devemos conhecer alguns elementos básicos que compõem um Sketch. Desta forma, a seguir apresentaremos a estrutura de um sketch, variáveis, funções e bibliotecas.

## Estruturas básica de um Sketch

Todos os Sketch Arduino devem ter a estrutura composta pelas funções `setup()` e `loop()` , conforme ilustra o exemplo a seguir.

|    |                           |
|----|---------------------------|
| 1  | <code>void setup()</code> |
| 2  | {                         |
| 3  | Comando 1;                |
| 4  | Comando 2;                |
| 5  | ...                       |
| 6  | }                         |
| 7  | <code>void loop()</code>  |
| 8  | {                         |
| 9  | Comando 3;                |
| 10 | Comando 4;                |
| 11 | ...                       |
| 12 | }                         |

A função `setup()` é executada apenas uma vez na inicialização do programa, e é nela que você deverá descrever as configurações e instruções gerais para preparar o programa antes que o loop principal seja executado. Em outras palavras, a função `setup()` é responsável pelas configurações iniciais da placa microcontroladora, tais como definições de pinos de entrada e saída, inicialização da comunicação serial, entre outras.

A função `loop()` é a função principal do programa e é executada continuamente enquanto a placa microcontroladora estiver ligada. É nesta função que todos os comandos e operações deverão ser escritos.

### **OBSERVAÇÕES:**

- As instruções dadas em um Sketch são realizadas de forma sequencial;
- É necessário incluir o sinal ; (ponto e vírgula) para que o programa identifique onde uma instrução deve ser finalizada.

# Variáveis

As variáveis são expressões que podemos utilizar em programas para nomear e armazenar um dado para uso posterior, como dados de um sensor ou um valor calculado.

Antes de serem utilizadas, todas as variáveis devem ser declaradas, definindo seu tipo e, opcionalmente, definindo seu valor inicial. Alguns tipos de variáveis encontram-se listadas na Tabela 2:

**Tabela 2** - Tipos de variáveis e suas descrições.

| Tipo   | Descrição                                                                  |
|--------|----------------------------------------------------------------------------|
| char   | Utilizado para armazenar um valor de caractere.                            |
| byte   | Usado para armazenar um número entre 0 e 255.                              |
| int    | Utilizado para armazenar números inteiros.                                 |
| bool   | Empregado para armazenar dois valores: true (verdadeiro) ou false (falso). |
| float  | Armazena números decimais que ocupam 32 bits (4 bytes).                    |
| double | Armazena números decimais que ocupam 64 bits (8 bytes).                    |
| void   | Usada apenas em declarações de funções                                     |
| String | Utilizada para armazenar cadeias de texto.                                 |

A declaração de uma variável pode ser melhor entendida com o exemplo a seguir. Neste caso, declaramos uma variável com nome a do tipo `int`, uma variável com nome b do tipo `float` e uma variável de nome C do tipo `char`. Quando as variáveis são declaradas antes da função `setup()`, significa que elas são variáveis globais, que podem ser usada em todas ao longo de todo o programa. Por sua vez, as variáveis declaradas em funções específicas, como no `loop()`, são variáveis locais e só podem ser usadas dentro da sua função de origem.

|   |                           |
|---|---------------------------|
| 1 | <code>int a;</code>       |
| 2 | <code>float b;</code>     |
| 3 | <code>char C;</code>      |
| 4 |                           |
| 5 | <code>void setup()</code> |
| 6 | <code>{</code>            |
| 7 | <code>...</code>          |
| 8 | <code>}</code>            |

Para definir um valor inicial a variável utilizamos o comando de atribuição, representado pelo símbolo `=`. Desta forma, quando escrevemos `int a = 10;` estamos atribuindo o valor 10 a variável de nome `a` do tipo `int`.

### **OBSERVAÇÕES:**

- O nome das variáveis deve seguir algumas regras: Devem iniciar com letras ou sublinhado `_` e não podem ter nome idêntico a alguma das palavras reservadas pelo programa ou biblioteca;
- Letras maiúsculas e minúsculas fazem diferença! Se o nome da variável definida for algo como, por exemplo, `ledPin`, não será a mesma coisa que `LedPin` ou `LEDPIN`;

O símbolo `=` tem o papel exclusivo de atribuição. A igualdade matemática é representada pela dupla igualdade `==`.



# Funções

Funções são blocos de instruções que podem ser chamados em qualquer parte do seu Sketch. A principal motivação para o uso de funções em um programa é quando necessitamos executar a mesma ação várias vezes. Desta forma, a segmentação do código em funções permite a criação de trechos de código modulares que executam uma tarefa definida e retornam à área do código a partir da qual a função foi chamada.

O uso de funções possui várias vantagens, entre elas:

- As funções ajudam o programador a manter o sketch organizado;
- As funções codificam uma ação em um só lugar, de forma que o trecho do código precise ser pensado e escrito apenas uma vez;
- Reduz as chances de erros na modificação quando o código precisa ser alterado;
- Facilitam a reutilização de código em outros programas;
- Tornam o código mais legível.

As duas funções principais na criação de um sketch no Arduino são `void setup()` e `void loop()`, mas existem algumas outras funções predefinidas para controlar uma placa microcontroladora, conforme mostra a Tabela 3:

**Tabela 3** – Funções e suas descrições.

| Controle                 | Função                      | Descrição                                  |
|--------------------------|-----------------------------|--------------------------------------------|
| Entrada e saída digitais | <code>digitalRead()</code>  | Lê o valor de um pino digital especificado |
|                          | <code>digitalWrite()</code> | Escreve no pino digital HIGH ou LOW        |

|                             |                                |                                                                                                     |
|-----------------------------|--------------------------------|-----------------------------------------------------------------------------------------------------|
|                             | <code>pinMode()</code>         | Configura o pino para funcionar como saída ou entrada                                               |
| Entrada e saída analógicas  | <code>analogRead()</code>      | Lê o valor de um pino analógico especificado                                                        |
|                             | <code>analogReference()</code> | Configura a tensão de referência para a entrada analógica                                           |
|                             | <code>analogWrite()</code>     | Aciona uma onda PWM em um pino                                                                      |
| Funções temporizadoras      | <code>delay()</code>           | Pausa o programa por um período (em milissegundos)                                                  |
|                             | <code>millis()</code>          | Retorna o número de milissegundos passados desde que a placa Arduino começou a executar o programa. |
| Entradas e saídas avançadas | <code>tone()</code>            | Gera uma onda quadrada na frequência especificada em um pino                                        |
|                             | <code>noTone()</code>          | Interrompe a função <code>tone()</code>                                                             |
| Funções matemáticas         | <code>map()</code>             | Mapeia um intervalo numérico em outro intervalo desejado                                            |
|                             | <code>sq()</code>              | Calcula o quadrado de um número                                                                     |
|                             | <code>sqrt()</code>            | Calcula a raiz quadrada de um número                                                                |
| Funções trigonométricas     | <code>cos()</code>             | Calcula o cosseno de um ângulo (em radianos)                                                        |
|                             | <code>sin()</code>             | Calcula o seno de um ângulo (em radianos)                                                           |
|                             | <code>tan()</code>             | Calcula a tangente de um ângulo (em radianos)                                                       |
| Números aleatórios          | <code>random()</code>          | Gera números pseudoaleatórios.                                                                      |
|                             | <code>randomSeed()</code>      | Inicializa o gerador de números pseudoaleatórios                                                    |
| Comunicação                 | <code>Serial</code>            | Usada para comunicação entre uma placa Arduino e um computador ou outros dispositivos               |
| Interrupções Externas       | <code>attachInterrupt()</code> | Cria interrupção externa                                                                            |
|                             | <code>detachInterrupt()</code> | Desativa a interrupção externa                                                                      |
| Interrupções                | <code>interrupts()</code>      | Reativa interrupções                                                                                |
|                             | <code>noInterrupts()</code>    | Desativa interrupções                                                                               |
| <b>OBSERVAÇÕES:</b>         |                                |                                                                                                     |

- Para saber mais sobre as funções predefinidas do Arduino acesse o site: <https://www.arduino.cc/reference/pt/>. Nele você encontrará outras funções não especificadas neste material de apoio e poderá consultar a descrição, sintaxe e parâmetros das funções que desejar;
- Os sinais analógicos são aqueles que variam continuamente ao longo do tempo. Por sua vez, os sinais digitais assumem valores discretos (0 ou 1);
- Configurar um pino digital em HIGH significa colocar o pino digital em nível lógico alto (1), ou seja 5 V. Definir um pino digital como LOW significa colocar o pino digital em nível lógico baixo (0), ou seja, 0 V.
- Outros conceitos técnicos descritos na Tabela 3 serão detalhados ao decorrer desse material.

Além destas funções, você também pode escrever suas próprias funções, que devem ser escritas fora das funções `setup()` e `loop()`. A criação de uma função deve seguir a sintaxe descrita abaixo:

|   |                                                            |
|---|------------------------------------------------------------|
| 1 | <code>tipo nome_da_funcao(declaração de parâmetros)</code> |
| 2 | <code>{</code>                                             |
| 3 | <code>Declaração de variável; //opcional</code>            |
| 4 | <code>Comando 1;</code>                                    |
| 5 | <code>Comando 1;</code>                                    |
| 6 | <code>...</code>                                           |
| 7 | <code>}</code>                                             |

Há dois tipos de função: As que não retornam nenhum valor e as que retornam algum valor para a função onde está inserida.

A função que não retornam nenhum valor são do tipo `void`. As funções do tipo `void` não retorna nenhum valor para a função que a chamou, ou seja, as ações executadas nessa função não retornam números, variáveis ou caracteres para a função principal.

Por sua vez, as funções que retornam valor podem ser do tipo `int`, `float`, `string`, etc. Uma função do tipo `int`, por exemplo, retorna um valor inteiro para a função que a chamou. Existem duas formas de retorno de uma função, uma delas é quando o finalizador de função ( `}` ) é encontrado e a outra é usando a declaração `return`.

A declaração `return` termina uma função e retorna um valor desejado.

### **OBSERVAÇÕES:**

- Se a função retorna um valor é obrigatório que seja determinado o tipo de retorno, que pode ser um número inteiro, um caractere ou um número real;
- As variáveis declaradas no parâmetro são variáveis de entrada, cujos tipos também devem ser especificados;
- Barra dupla ( `//` ) pode ser utilizada para fazer um breve comentário em alguma linha do código. A função do comentário é deixar o código claro tanto para o programador quanto para outras pessoas. Os comentários são ignorados pelo compilador do código;
- Também é possível comentar várias linhas do código, para isso você deve incluir os comandos `/*` na linha de início e `*/` ao final da linha que finaliza o trecho a ser comentado.

# Bibliotecas

As bibliotecas são um conjunto de instruções desenvolvidas para executar tarefas específicas relacionadas a um determinado dispositivo. O uso de bibliotecas facilita a conexão a sensores, a uma tela, a um módulo, etc., além de poupar tempo do programador.

Algumas bibliotecas já vêm instaladas com o Arduino IDE, outras podem ser incluídas a partir de download e você também pode criar a sua própria. Muitas vezes o fabricante de um sensor, módulos, atuador, etc. fornece uma biblioteca para facilitar a programação.

Uma biblioteca padrão é chamada através da seguinte instrução:

|   |                                                  |
|---|--------------------------------------------------|
| 1 | <code>#include &lt;nome_da_biblioteca&gt;</code> |
|---|--------------------------------------------------|

Por sua vez, uma biblioteca criada pelo usuário segue a sintaxe:

|   |                                              |
|---|----------------------------------------------|
| 1 | <code>#include "nome_da_biblioteca.h"</code> |
|---|----------------------------------------------|

## OBSERVAÇÕES:

- Usamos a diretiva `#include` quando desejamos incluir uma biblioteca externa ao nosso Sketch. Com isso, teremos acesso a um grande número de bibliotecas escritas especialmente para a linguagem Arduino;
- Outra diretiva que será utilizada em nossos projetos é a `#define`, que permite dar um nome a um valor constante antes de o programa ser compilado. Uma vantagem da utilização desta diretiva é que as variáveis definidas a partir dela não ocupam espaço na memória de programa do chip;

- Instruções com `#include` e `#define` não são terminadas com ; (ponto e vírgula).

# Operadores

## Operadores aritméticos

Os operadores aritméticos são as representações que utilizamos para realizar as operações aritméticas básicas, como somar, subtrair, dividir, multiplicar, entre outras.

**Tabela 4 - Operadores aritméticos.**

| Operador | Nome             | Sintaxe      | Resultado                                |
|----------|------------------|--------------|------------------------------------------|
| +        | Adição           | $x = y + z$  | x é igual a soma de y mais z             |
| -        | Subtração        | $x = y - z$  | x é igual a subtração de y menos z       |
| *        | Multiplicação    | $x = y * z$  | x é igual a multiplicação de y vezes z   |
| /        | Divisão          | $x = y / z$  | x igual a divisão de y por z             |
| %        | Resto da divisão | $x = y \% z$ | x é igual ao resto da divisão de y por z |

## Operadores de comparação e booleanos

Para programas corretamente a placa microcontroladora UNO é necessário aprender a usar de forma adequada os operadores de comparação e booleanos.

Os operadores de comparação, como o próprio nome diz, compara dois valores retornando verdadeiro (true) ou falso (false). Observe na Tabela 5 a seguir os operadores de comparação.

**Tabela 5 - Operadores de comparação.**

| Operador | Nome | Sintaxe | Resultado |
|----------|------|---------|-----------|
|----------|------|---------|-----------|

|    |                  |        |                                                       |
|----|------------------|--------|-------------------------------------------------------|
| == | Igual            | x == y | Retorna true (verdadeiro) se x for igual a y          |
| != | Diferente        | x != y | Retorna true (verdadeiro) se x for diferente de y     |
| <  | Menor que        | x < y  | Retorna true (verdadeiro) se x for menor que y        |
| >  | Maior que        | x > y  | Retorna true (verdadeiro) se x for maior que y        |
| <= | Menor ou igual a | x <= y | Retorna true (verdadeiro) se x for menor ou igual a y |
| >= | Maior ou igual a | x >= y | Retorna true (verdadeiro) se x for maior ou igual a y |

Os operadores booleanos são utilizados para testes lógicos entre elementos em um teste condicional. Assim como os operadores de comparação, os operadores booleanos também retornam verdadeiro (true) e falso (false) conforme o resultado dos testes. Os operadores booleanos são:

**Tabela 6** - Operadores booleanos.

| Operador | Nome       |
|----------|------------|
| &&       | E lógico   |
|          | OU lógico  |
| !        | NÃO lógico |

### Operador E lógico

O E lógico resulta em verdadeiro apenas se **ambos** os operandos forem verdadeiros. Representamos o E lógico com &&. Exemplo: Se quisermos verificar se um determinado valor de temperatura se encontra entre uma faixa de valores (entre 30 e 50°C), podemos utilizar:



Se temperatura >=30 && temperatura <=50

Se o valor encontrado de temperatura for maior ou igual a 30 e menor ou igual a 50 a condição será satisfeita, retornando verdadeiro (true).

## Operador OU lógico

O OU lógico resulta em verdadeiro se **pelo menos um** dos operadores for verdadeiro. Representamos o OU lógico com ||.

Exemplo: Se quisermos verificar se o valor de temperatura é igual a pelo menos um valor determinado (30°C ou 50°C), podemos utilizar:

Se temperatura == 30 || temperatura == 50

Neste caso, se o valor encontrado de temperatura for igual a 30 **ou** igual a 50 a condição será satisfeita, retornando verdadeiro (true).

## Operador NÃO lógico

O NÃO lógico resulta em verdadeiro se o operando for falso. Representamos o NÃO lógico com !.

Exemplo: Se quisermos mudar o estado de uma variável x de verdadeiro para falso e vice-versa, podemos utilizar:

!x

Neste caso, se x for verdadeiro (true) o uso do NÃO lógico transformará seu estado para falso (false). De mesmo modo, se x for falso (false) o NÃO lógico transformará seu estado para verdadeiro (true).

## Operadores de incremento e decremento

Os operadores de incremento e decremento são operadores unitários utilizados com a finalidade de adicionar ou subtrair em uma unidade ao valor de uma variável. O operador de incremento é representado por ++ e o de decremento por --.

Os operadores de incremento e decremento podem ser pré-fixos ou pós-fixos dependendo de serem posicionados antes ou depois do nome da variável a ser implementada ou decrementada.

**Tabela 7** - Operadores de incremento e decremento.

| Tipo      | Operador | Significado                                      |
|-----------|----------|--------------------------------------------------|
| Prefixo   | ++x      | Incrementa x em um e retorna o novo valor de x   |
|           | --x      | Decrementa x em um e retorna o novo valor de x   |
| Pós-fixos | x++      | Incrementa x em um e retorna o valor antigo de x |
|           | x--      | Decrementa x em um e retorna o valor antigo de x |

Exemplo:

|   |                                      |
|---|--------------------------------------|
| 1 | x=2;                                 |
| 2 | y=++x; //x agora contém 3 e y também |

|   |                                                        |
|---|--------------------------------------------------------|
| 3 | <code>y=x++;</code> //x contém 4, mas y ainda contém 3 |
|---|--------------------------------------------------------|

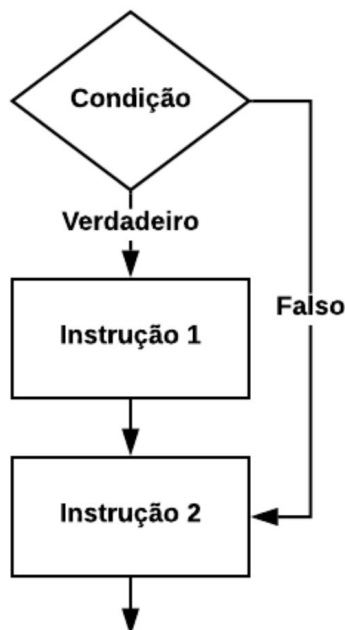
## Estrutura de controle de fluxo

As estruturas de controle são blocos de programação que analisam variáveis e decidem como o programa deve se comportar com base em parâmetros pré-definidos. Em outras palavras, a estrutura de controle de fluxo determina como o programa responderá diante de certas condições ou parâmetros.

A Figura 22 ilustra um exemplo de estrutura de controle, em que dependendo do resultado obtido na condicional (verdadeiro ou falso) o programa executará diferentes instruções.

Neste caso, o resultado da condicional for verdadeiro o programa executará a Instrução 1 e, em sequência, a instrução 2. No entanto, se o resultado da condicional for falso apenas a Instrução 2 será executada pelo programa.

**Figura 22** - Estrutura de controle de fluxo.



## IF

O comando `if` (se) é uma estrutura de controle de fluxo de seleção. Usamos esse comando para checar uma condição. Se a condição for satisfeita (verdadeiro) uma lista de instrução delimitada por `{}` (chaves) serão executadas. No entanto, se a condição não for satisfeita (falso) esta lista de instruções não será executada e o programa seguirá a sequência de comandos escritos depois do `if`.

A sintaxe do comando `if` na programação Arduino é:

|   |                             |
|---|-----------------------------|
| 1 | <code>if (condição){</code> |
| 2 | <code>  Comando 1;</code>   |
| 3 | <code>  Comando 2;</code>   |
| 4 | <code>  ... }</code>        |

## IF...ELSE

A combinação dos comandos *if...else* permite maior controle sobre o fluxo de código que o comando *if*, por possibilitar que múltiplos testes sejam agrupados. O comando *else* (senão) será executado se a condição do comando *if* (se) resultar em falso.

A sintaxe dos comandos *if...else* na programação Arduino é:

|   |                              |
|---|------------------------------|
| 1 | <code>if (condição1){</code> |
| 2 | <code>  Comando 1;</code>    |
| 3 | <code>}</code>               |
| 4 | <code>else{</code>           |
| 5 | <code>  Comando 2;</code>    |
| 6 | <code>}</code>               |

Dentro do comando *else* podemos adicionar outro comando *if*, de modo que múltiplos testes podem ser executados ao mesmo tempo. Desta forma, a sintaxe poderá ser a seguinte:

|   |                             |
|---|-----------------------------|
| 1 | <i>if</i> (condição1){      |
| 2 | Comando 1;                  |
| 3 | }                           |
| 4 | <i>else if</i> (condição2){ |
| 5 | Comando 2;                  |
| 6 | }                           |
| 7 | <i>else</i> {               |
| 8 | Comando 3;                  |
| 9 | }                           |

## FOR

O comando *for* (para) é uma estrutura de controle de fluxo de repetição. Este comando permite que certo trecho do código seja executado um determinado número de vezes. O comando *for* é útil para qualquer operação repetitiva e é usado frequentemente com vetores para operar em coleções de dados.

A sintaxe do comando *for* é a seguinte:

|   |                                                   |
|---|---------------------------------------------------|
| 1 | <i>for</i> (inicialização; condição; incremento){ |
| 2 | Comando 1;                                        |
| 3 | ...                                               |
| 4 | }                                                 |

A inicialização ocorre primeiro e apenas uma vez. A cada repetição do loop, a condição é testada, se verdadeira o bloco de

comandos e o incremento são executados. Por sua vez, quando a condição for falsa o loop termina.

## SWITCH...CASE

Da mesma forma que o comando *if*, o comando *switch...case* (selecione...caso) é uma estrutura de controle de fluxo de seleção. Este comando permite especificar códigos diferentes para serem executados em várias condições, funcionando da seguinte maneira: um comando *switch* compara o valor de uma variável aos valores especificados nos comandos *case*. Quando um comando *case* é encontrado, cujo valor é igual ao da variável, o código para esse comando *case* é executado.

A sintaxe do comando *switch...case* é a seguinte:

|    |                                |
|----|--------------------------------|
| 1  | <code>switch (var){</code>     |
| 2  | <code>    case valor1:</code>  |
| 3  | <code>        comando1;</code> |
| 4  | <code>    break;</code>        |
| 5  | <code>    case valor2:</code>  |
| 6  | <code>        comando2;</code> |
| 7  | <code>    break;</code>        |
| 8  | <code>    default:</code>      |
| 9  | <code>        comando3;</code> |
| 10 | <code>    break;</code>        |
| 11 | <code>}</code>                 |

A variável `var` será comparada ao valor dos vários cases, podendo ser do tipo `int` ou `char`. Os parâmetros `valor1` e `valor2` são constantes do tipo `int` ou `char`.

### **OBSERVAÇÕES:**

- A palavra-chave *break* é utilizada para interromper o comando *switch*, devendo ser escrito ao final de cada *case*. Sem o comando *break* o comando *switch* continuará executando as expressões seguintes, até encontrar um *break*.
- A palavra-chave *default* é opcional e é executado quando a variável de comparação do *switch* não corresponde a nenhum valor constate dos *cases*.

### **WHILE**

O *while* (enquanto) é uma estrutura de controle de fluxo de repetição. As instruções contidas em um *while* serão repetidas continuamente, e infinitamente, até que a sua condicional se torne falsa. Em outras palavras, as instruções contidas no código *while* serão executadas enquanto a condição for satisfeita (verdadeiro)

A forma geral do *while* é:

|   |                          |
|---|--------------------------|
| 1 | <i>while</i> (condição){ |
| 2 | Comando 1;               |
| 3 | ...                      |
| 4 | }                        |

### **DO...WHILE**



A estrutura *do...while* (faça...enquanto) funciona de forma semelhante às estruturas *while* e *for*, com exceção de a condição ser testada ao final do loop, de tal modo que as instruções contidas no *do...while* serão executadas pelo menos uma vez.

A sintaxe do comando *do...while* é a seguinte:

|   |                     |
|---|---------------------|
| 1 | do{                 |
| 2 | Comando 1;          |
| 3 | } while (condição); |





# CRIANDO JOGOS COM ARDUINO

# STOP – UM JOGO COM LEDS

O STOP é um jogo simples feito com LEDs. Neste jogo, os LEDs ligam e desligam um após o outro e o jogador deve atingir o objetivo de pressionar o botão no exato momento em que o LED do meio estiver aceso. À medida que o jogador consegue atingir o objetivo ele passará de nível e os LEDs piscam com maior velocidade.

## MATERIAIS NECESSÁRIOS

- 1 x [Placa UNO SMD R3 Atmega328 compatível com Arduino UNO](#);
- 1 x [Cabo USB](#);
- 1 x [Protoboard](#);
- 11 x [LEDs difusos de 5 mm](#);
- 11 x [Resistores de 220  \$\Omega\$](#) ;
- 1 x [Resistores de 10 k \$\Omega\$](#) ;
- 1 x [Botões do tipo push button](#);

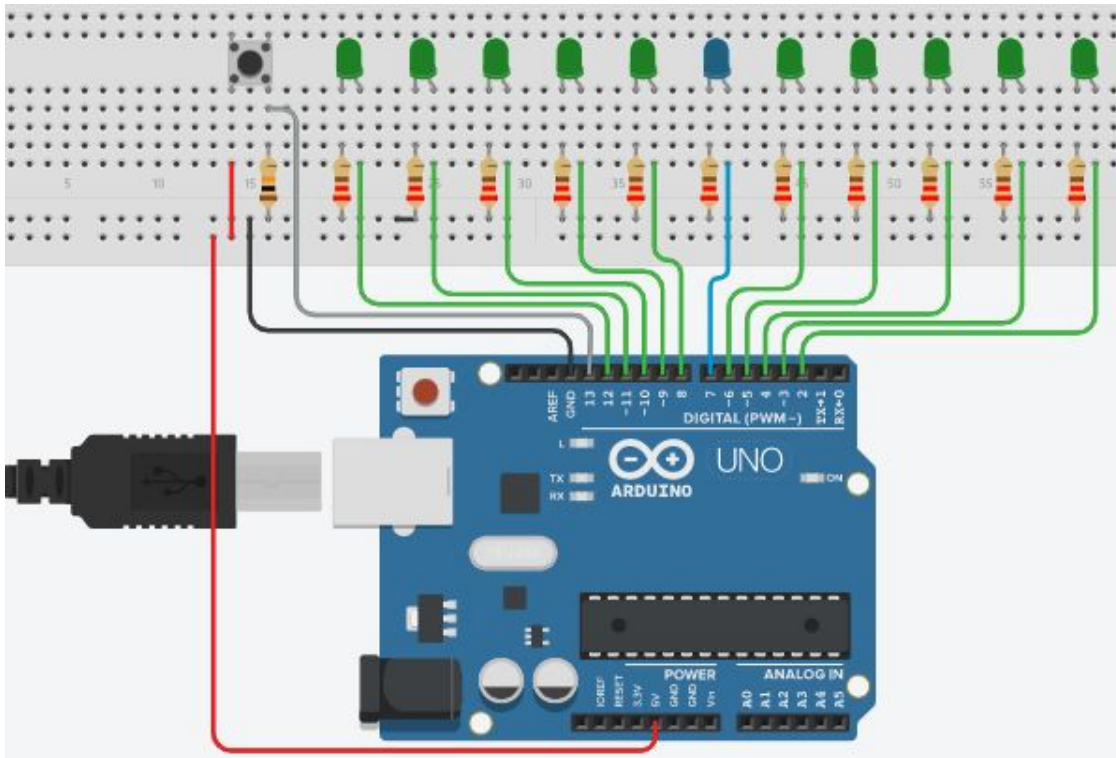
## ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

Monte o circuito conforme a Figura 23 utilizando a protoboard, os LEDs, o botão, os resistores e os fios.

Ao montar o circuito na protoboard preste atenção nos seguintes pontos:

- Os terminais mais longos dos LEDs encontram-se conectados aos pinos Digitais de 2 a 13. Estes terminais longos são os ânodos (positivos) dos LEDs e devem ser conectados na alimentação de 5V, neste caso representado pelos pinos digitais. Os terminais mais curtos são os cátodos (negativos) e devem ser conectados ao terra (GND);
- Vamos conectar o botão com um resistor de 10 k $\Omega$  pull-down (resistor conectado no GND). Desta forma, quando o botão estiver pressionado a placa UNO retornará nível lógico alto (HIGH) no pino digital 13.

**Figura 23** - Circuito para STOP – Um Jogo com LEDs.



ELABORANDO O CÓDIGO

Com o circuito montado vamos a programação do jogo. Para melhor compreensão o código será explicado passo-a-passo a seguir. Neste momento, observe o código abaixo e aproveite para analisar sua estrutura.

```
int tempo = 600; //Tempo de acionamento do LED
int nivel = 0; // Variável para armazenar o nível do jogador

void setup()
{
  Serial.begin(9600); //Inicialização da comunicação serial
  pinMode(13, INPUT); //Define o pino digital 13 (botão) como entrada

  //Configura os pinos em que os LEDs estão conectados como saída
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(2, OUTPUT);

  Serial.print("Nível -> "); //Imprime na serial "Nível -> "
  Serial.println(nivel); //Imprime na serial o valor de nível
}

void loop()
{
  for (int x = 2; x <= 12; x++) { //Inicializa x = 2, enquanto x for menor ou igual a 12,
incrementa x em 1
    digitalWrite(x, HIGH); //Coloca o LED x em nível alto
    delay(tempo); //Intervalo de 600 milissegundos (tempo)

    if (digitalRead(13) == 1) { //Se a leitura do pino 13 (botão) for igual a 1
      while (digitalRead(13)); //Enquanto o botão estiver pressionado o jogo fica
travado
    }
  }
}
```

```

    if (x == 7) { //Se x for igual a 7 (pino do led azul)
        nivel = nivel + 1; //Acrescenta nivel em 1
        Serial.print("Nivel -> "); //Imprime na serial "Nivel -> "
        Serial.println(nivel); //Imprime na serial o valor de nível
        tempo = tempo - 100; //Diminui tempo em 100 milissegundos
    } else { //Senão
        Serial.println("GAME OVER"); //Imprime na serial "GAME OVER"
        game_over(); //Chama a função game_over()
        tempo = 600; //Retorna tempo = 600
        x = 2; //Retorna x =2
        nivel = 0; //Retorna nivel = 0
        Serial.print("Nivel -> "); //Imprime na serial "Nivel -> "
        Serial.println(nivel); //Imprime na serial o valor de nível
    }
}

digitalWrite(x, LOW); //Coloca o LED x em nível baixo
}

for (int x = 12; x >= 2; x--) { //Inicializa x = 2, enquanto x for maior ou igual a 12,
decrementa x em 1
    digitalWrite(x, HIGH); //Coloca o LED x em nível alto
    delay(tempo); //Intervalo de 600 milissegundos (tempo)
    digitalWrite(x, LOW); //Coloca o LED x em nível baixo

    if (digitalRead(13) == 1) { //Se a leitura do pino 13 (botão) for igual a 1
        while (digitalRead(13)); //Enquanto o botão estiver pressionado o jogo fica
travado
        if (x == 7) { //Se x for igual a 7 (pino do led azul)
            nivel = nivel + 1; //Acrescenta nivel em 1
            Serial.print("Nivel -> "); //Imprime na serial "Nivel -> "
            Serial.println(nivel); //Imprime na serial o valor de nível
            tempo = tempo - 100; //Diminui tempo em 100 milissegundos
        } else { //Senão
            Serial.println("GAME OVER"); //Imprime na serial "GAME OVER"
            game_over(); //Imprime na serial "GAME OVER"
            tempo = 600; //Retorna tempo = 600
            x = 12; //Retorna x = 12
            nivel = 0; //Retorna nível = 0
            Serial.print("Nivel -> "); //Imprime na serial "Nivel -> "
            Serial.println(nivel); //Imprime na serial o valor de nível
        }
    }
}

```



```

    digitalWrite(x, LOW); //Coloca o LED x em nível baixo
}
}
//Função game_over()
void game_over()
{
    for (int j = 0; j < 4; j++) //Inicializa j = 0, enquanto j for menor que 4, decrementa j
em 1
        for (int x = 2; x <= 12; x++) { //Inicializa x = 2, enquanto x for maior ou igual a 12,
decrementa x em 1
            digitalWrite(x, HIGH); //Coloca o LED x em nível alto
            digitalWrite(14 - x, HIGH); //Coloca o LED 14 - x em nível alto
            delay(100); //Intervalo de 100 ms
            digitalWrite(x, LOW); //Coloca o LED x em nível baixo
            digitalWrite(14 - x, LOW); //Coloca o LED 14 - x em nível baixo
        }
}
}

```

Para melhor compreensão do código acompanhe os seguintes passos:

#### 1- Declarar as variáveis

Neste código declaramos duas variáveis: `tempo`, inicializada em 600, para determinar o tempo de acionamento dos LEDs; e `nível` para armazenar o nível que o jogador está.

#### 2- Inicialização da comunicação serial e configuração das portas de entrada e saída

A comunicação serial foi inicializada por meio da instrução: `Serial.begin(9600);`.

O pino digital 13, em que o botão está conectado, foi configurado como entrada ( `INPUT` ) e os demais (pinos de 2 a 12) foram configurados como saída ( `OUTPUT` ).

#### 3- Ligar e desligar os LEDs sequencialmente – Ordem crescente

Iniciamos o `loop()` com a estrutura de repetição *for* para fazer o acionamento sequencial dos LEDs, iniciando do LED da porta digital 2 e finalizando no LED da porta digital 12.

#### 4- Realizar a leitura do botão

Em seguida, realizamos a leitura do botão e verificamos se o valor lido é igual a 7, porta digital em que o LED azul encontra-se conectado. Caso esta condição seja verdadeira, o jogador conseguiu atingir o LED azul e o nível do jogo será aumentado. Senão, o jogador não alcançou o LED azul, perdendo o jogo.

#### 5- Ligar e desligar os LEDs sequencialmente – Ordem decrescente

De forma semelhante à etapa 3, usaremos a estrutura de repetição *for* para ligar e desligar os LEDs sequencialmente, iniciando do LED da porta 12 e finalizando no LED da porta 2.

#### 6- Realizar a leitura do botão

Novamente realizamos a leitura do botão para verificar se o jogador atingiu o LED 7.

#### 7- Função Game Over

A função `gameover()` foi criada para criar um efeito nos LEDs quando o jogador perde o jogo. Assim, quando o jogador não consegue alcançar o LED azul esta função é chamada.

## TINKERCAD

Para melhor visualizar e/ou simular o circuito e a programação deste projeto basta se acessar o seguinte link: <http://blogdarobotica.com/tinkercad-STOP>.



# JOGO DA MEMÓRIA

O jogo eletrônico Genius foi um brinquedo muito popular na década de 1980. Esse jogo busca estimular a memorização através de sequências de cores e sons. Para vencer, o jogador deve reproduzir essas sequências na mesma ordem em que lhe é apresentada.

A proposta desse projeto é criar um jogo de memória similar ao game Genius utilizando componentes eletrônicos em conjunto com a placa UNO. Neste projeto, reuniremos todos os conceitos básicos aprendidos até aqui para nos divertir. Além disso, vamos aprender como produzir uma sequência de números randômicos.

Para criar as sequências de cores e sons do jogo utilizaremos as funções `randomSeed()` e `random()`.

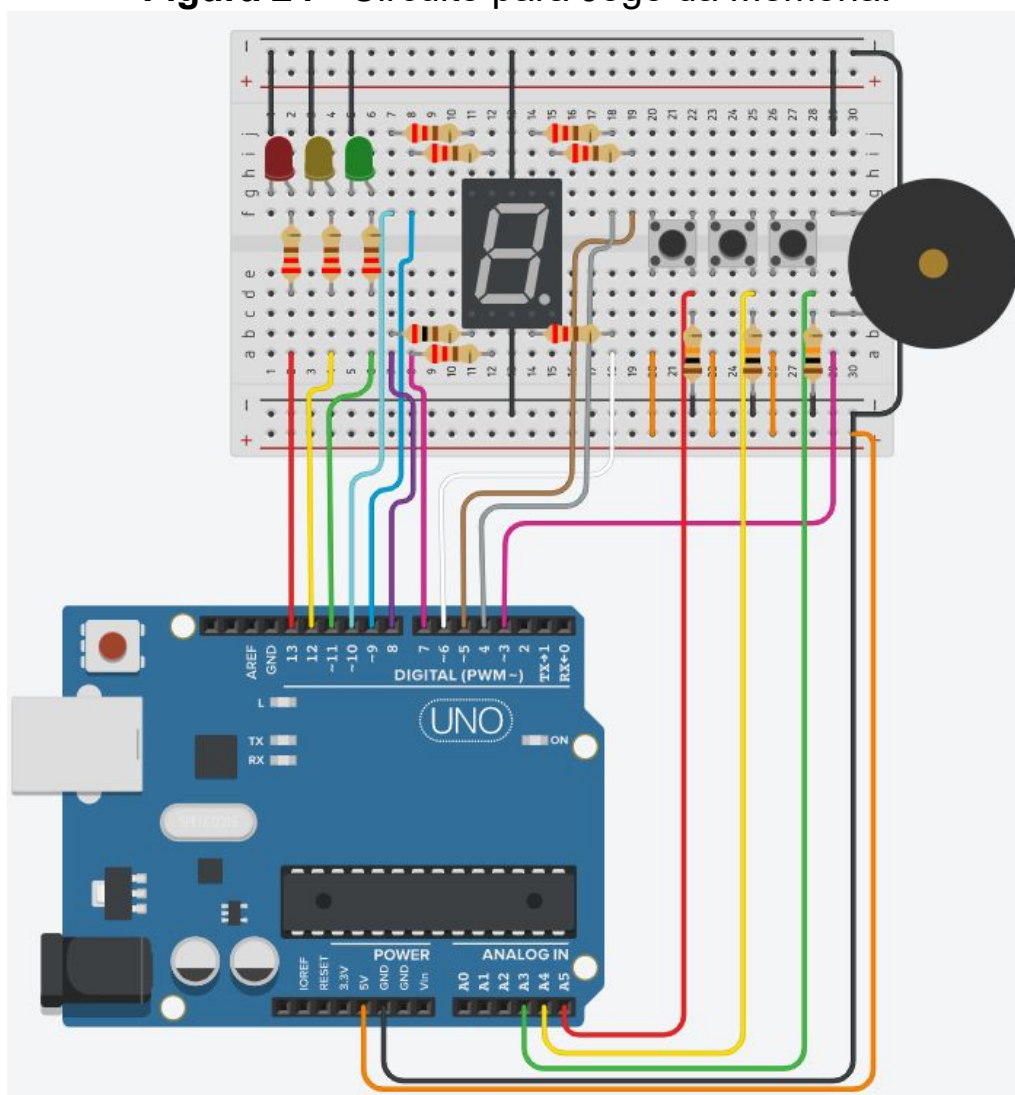
## MATERIAIS NECESÁRIOS

- 1 x [Placa UNO SMD R3 Atmega328 compatível com Arduino UNO](#);
- 1 x [Cabo USB](#);
- 1 x [Protoboard](#);
- 3 x [LEDs difusos de 5 mm](#);
- 1 x [Display de 7 segmentos cátodo comum](#);
- 9 x [Resistores de 220  \$\Omega\$](#) ;
- 3 x [Resistores de 10 k \$\Omega\$](#) ;
- 1 x [Buzzer ativo](#);
- 3 x [Botões do tipo push button](#);

## ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

Conecte os componentes na protoboard como ilustra a Figura 24. Verifique cuidadosamente os cabos de ligação antes de ligar sua placa UNO. Lembre-se que a placa UNO deve estar desconectada enquanto você monta o circuito.

**Figura 24 - Circuito para Jogo da Memória.**



Ao montar seu circuito na protoboard preste atenção nos seguintes pontos:

- Construiremos o Game Genius utilizando apenas três sequências de cores e sons. Desta forma, utilizaremos três LEDs (vermelho, amarelo e verde) e três botões que representaram as cores dos LEDs e que devem ser pressionados pelo usuário para reproduzir a sequência;
- O LED vermelho deve ser conectado à porta digital 13, o amarelo à porta digital 12 e o verde à porta digital 11;
- Utilizaremos o display de 7 segmentos para informar ao jogador em qual nível do jogo ele está. O display de 7 segmentos é do tipo cátodo comum, então os terminais comuns do mesmo devem ser conectados ao terra (GND);
- Os segmentos encontram-se conectado as portas digitais da seguinte maneira: a = 4; b = 5; c = 6; d = 7; e = 8; f = 9; g = 10;
- O buzzer ativo deve ser conectado à porta digital 3;
- O primeiro botão, correspondente ao botão vermelho, será utilizado para acionar o LED vermelho e deve ser conectado a uma resistência pull-down e a porta analógica A5;
- O segundo botão, correspondente ao botão amarelo, será utilizado para acionar o LED amarelo e deve ser conectado a uma resistência pull-down e a porta analógica A4;
- O último botão, correspondente ao botão verde, será utilizado para acionar o LED verde e deve ser conectado a uma resistência pull-down e a porta analógica A3.

## ELABORANDO O CÓDIGO

Com o circuito montado vamos a programação do Sketch do Jogo da Memória. Para melhor compreensão o código será explicado passo-a-passo a seguir. Neste momento, observe o código abaixo e aproveite para analisar sua estrutura.

```
//ENTRADAS
int butRed = A5;//Atribui a porta A5 a variável butRed - botão vermelho
int butYel = A4;//Atribui a porta A4 a variável butYel - botão amarelo
int butGre = A3;//Atribui a porta A3 a variável butGre - botão verde

//SAÍDAS
//Led
int ledRed = 13;//Atribui a porta 13 a variável ledRed - LED vermelho
int ledYel = 12;//Atribui a porta 12 a variável ledYel - LED amarelo
int ledGre = 11;//Atribui a porta 11 a variável ledGre - LED verde
//Display
int a = 4;//Atribui a porta 4 a variável "a" - segmento a do display
int b = 5;//Atribui a porta 5 a variável "b" - segmento b do display
int c = 6;//Atribui a porta 6 a variável "c" - segmento c do display
int d = 7;//Atribui a porta 7 a variável "d" - segmento d do display
int e = 8;//Atribui a porta 8 a variável "e" - segmento e do display
int f = 9;//Atribui a porta 9 a variável "f" - segmento f do display
int g = 10;//Atribui a porta 10 a variável "g" - segmento g do display
//Buzzer
int buzzerPin = 3;//Atribui a porta 3 a variável buzzerPin - buzzer

//Tons botões
#define tomRed 294//Tom para LED vermelho
#define tomYel 330//Tom para LED amarelo
#define tomGre 350//Tom para LED verde

//VARIÁVEIS
int rodada = 0;//Variável que armazenará o número de rodadas
int led = 0;//Variável que armazenará o valor randômico que representará o LED que estará ligado
int temp_led_ligado = 600;//Tempo em que o LED deve estar ligado
int temp_led_desligado = 100;//Tempo em que o LED deve estar desligado
int mat[10];//Vetor de 10 posições que armazenará a sequência
int posi_mat = 0;//Posição do vetor
```

```

void setup(){
  //Configuração dos pinos de entrada
  pinMode(butRed, INPUT);
  pinMode(butYel, INPUT);
  pinMode(butGre, INPUT);
  //Configuração dos pinos de saída
  pinMode(ledRed, OUTPUT);
  pinMode(ledYel, OUTPUT);
  pinMode(ledGre, OUTPUT);
  pinMode(a, OUTPUT);
  pinMode(b, OUTPUT);
  pinMode(c, OUTPUT);
  pinMode(d, OUTPUT);
  pinMode(e, OUTPUT);
  pinMode(f, OUTPUT);
  pinMode(g, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
  //Inicialização da comunicação serial
  Serial.begin(9600);
  //Função randômica
  randomSeed(analogRead(A0)); //Pega como referência um valor aleatório do
  canal analógico para gerar os valores randômicos
  inicio(); //Chama a função início
}

void loop(){
  led = random(0, 3); //Gera o valor aleatório entre 0 e 2
  mat[posi_mat] = led; //Guarda o valor gerado no vetor

  switch (posi_mat) { //Liga o display conforme posição do vetor (nível do jogo)
    case 0:
      zero(); //Executa a função zero
      break;
    case 1:
      um(); //Executa a função um
      break;
    case 2:
      dois(); //Executa a função dois
      break;
    case 3:
      tres(); //Executa a função três
      break;
    case 4:
      quatro(); //Executa a função quatro
  }
}

```



```

    break;
case 5:
    cinco();//Executa a função cinco
    break;
case 6:
    seis();//Executa a função seis
    break;
case 7:
    sete();//Executa a função sete
    break;
case 8:
    oito();//Executa a função oito
    break;
case 9:
    nove();//Executa a função nove
    break;
}

```

for (int j = 0; j <= posi\_mat; j++) { //Mostra a sequência dos LEDs sempre que passar o nível

switch (mat[j]) { //Liga o LED correspondente a valor que está no vetor na posição j

```

case 0:
    digitalWrite(ledGre, HIGH); //Liga LED verde
    analogWrite(buzzerPin, tomGre); //Emite tom para LED verde
    delay(200); //Intervalo de 200 milissegundos
    digitalWrite(buzzerPin, LOW); //Desliga buzzer
    delay(temp_led_ligado);
    digitalWrite(ledGre, LOW); //Desliga LED verde
    delay (temp_led_desligado);
    break;
case 1:
    digitalWrite(ledYel, HIGH); //Liga LED amarelo
    analogWrite(buzzerPin, tomYel); //Emite tom para LED amarelo
    delay(200); //Intervalo de 200 milissegundos
    digitalWrite(buzzerPin, LOW); //Desliga buzzer
    delay(temp_led_ligado);
    digitalWrite(ledYel, LOW); //Desliga LED amarelo
    delay (temp_led_desligado);
    break;
case 2:
    digitalWrite(ledRed, HIGH); //Liga LED vermelho
    analogWrite(buzzerPin, tomRed); //Emite tom para LED vermelho

```

```

    delay(200); //Intervalo de 200 milissegundos
    digitalWrite(buzzerPin, LOW); //Desliga buzzer
    delay(temp_led_ligado);
    digitalWrite(ledRed, LOW); //Desliga LED vermelho
    delay (temp_led_desligado);
    break;
default:
    break;
}
}
int temp = 0; //Variável temporária utilizada para aguardar o jogador digitar a
sequência fornecida pelo jogo
while (temp <= posi_mat){
    if (digitalRead(butGre) == HIGH) { //Se butGre for nível alto - botão verde
pressionado
        digitalWrite(ledGre, HIGH); //Liga LED verde
        analogWrite(buzzerPin, tomGre); //Emite tom para LED verde
        delay(100); //Intervalo de 100 milissegundos
        digitalWrite(buzzerPin, LOW); //Desliga buzzer
        while (digitalRead(butGre) == HIGH); //Enquanto o botão verde estiver
pressionado não faz nada
        digitalWrite(ledGre, LOW); //Desliga o LED verde
        delay(200); //Intervalo de 200 milissegundos
        if ( mat[temp] == 0) { //Se na posição do vetor é verdadeiro (igual a zero)
            temp = temp + 1; //Incrementa a variável temp para dar continuidade na
sequência do jogo
        } else //Se não
        {
            Serial.println("ERRO!!!!"); //Imprime na serial Erro
            posi_mat = 11; //Atribui 11 na variável posi_mat
            break;
        }
    }
    if (digitalRead(butYel) == HIGH) { //Se butYel for nível alto - botão amarelo
pressionado
        digitalWrite(ledYel, HIGH); //Liga led amarelo
        analogWrite(buzzerPin, tomYel); //Emite tom para LED amarelo
        delay(100); //Intervalo de 100 milissegundos
        digitalWrite(buzzerPin, LOW); //Desliga buzzer
        while (digitalRead(butYel) == HIGH); //Enquanto o botão amarelo estiver
pressionado não faz nada
        digitalWrite(ledYel, LOW);
        delay(200);
        if ( mat[temp] == 1) { //Se na posição do vetor é verdadeiro (igual a zero)

```

```

    temp = temp + 1;
} else
{
    Serial.println("ERRO!!!!");
    posi_mat = 11; //Atribui 11 na variável posi_mat
    break;
}
}
if (digitalRead(butRed) == HIGH) { //Se butRed for nível alto - botão vermelho
pressionado
    digitalWrite(ledRed, HIGH); //Liga led vermelho
    analogWrite(buzzerPin, tomRed); //Emite tom para led vermelho
    delay(100); //Intervalo de 100 milissegundos
    digitalWrite(buzzerPin, LOW); //Desliga buzzer
    while (digitalRead(butRed) == HIGH);
    digitalWrite(ledRed, LOW);
    delay(200);
    if (mat[temp] == 2) {
        temp = temp + 1;
    } else
    {
        Serial.println("ERRO!!!!"); //Imprime na serial "ERRO!!!!"
        posi_mat = 11; //Atribui 11 na variável posi_mat
        break;
    }
}
}
posi_mat = posi_mat + 1;

if (posi_mat == 10) { //Se a posi_mat for igual a 10
ganhou(); //Chama a função ganhou()
    Serial.println("Parabens voce venceu "); //Imprime no serial "Parabens voce
venceu"
    posi_mat = 0; //Atribui 0 a variável posi_mat para reiniciar o jogo
    Serial.println("INICIO DE JOGO"); //Imprime no serial "INICIO DE JOGO"
}

if (posi_mat >= 11) { //Se posi_mat for maior ou igual a 11
    Serial.println(" ---- >>> FIM DE JOGO <<< ----"); //Imprime no serial " ---- >>>
FIM DE JOGO <<< ----"
    perdeu(); //Chama a função perdeu
    posi_mat = 0; //Atribui 0 a variável posi_mat para reiniciar o jogo
    Serial.println("INICIO DE JOGO"); //Imprime no serial "INICIO DE JOGO"
    delay(3000); //Intervalo de 3 segundos

```

```
}  
}
```

//Função para escrever os números no display

```
void zero() {  
    digitalWrite(a, 1);  
    digitalWrite(b, 1);  
    digitalWrite(c, 1);  
    digitalWrite(d, 1);  
    digitalWrite(e, 1);  
    digitalWrite(f, 1);  
    digitalWrite(g, 0);  
    delay(100);  
}
```

```
void um() {  
    digitalWrite(a, 0);  
    digitalWrite(b, 1);  
    digitalWrite(c, 1);  
    digitalWrite(d, 0);  
    digitalWrite(e, 0);  
    digitalWrite(f, 0);  
    digitalWrite(g, 0);  
    delay(100);  
}
```

```
void dois() {  
    digitalWrite(a, 1);  
    digitalWrite(b, 1);  
    digitalWrite(c, 0);  
    digitalWrite(d, 1);  
    digitalWrite(e, 1);  
    digitalWrite(f, 0);  
    digitalWrite(g, 1);  
    delay(100);  
}
```

```
void tres() {  
    digitalWrite(a, 1);  
    digitalWrite(b, 1);  
    digitalWrite(c, 1);  
    digitalWrite(d, 1);  
    digitalWrite(e, 0);  
    digitalWrite(f, 0);  
    digitalWrite(g, 1);  
    delay(100);  
}
```

```
void quatro() {  
    digitalWrite(a, 0);  
    digitalWrite(b, 1);  
    digitalWrite(c, 1);  
    digitalWrite(d, 0);  
    digitalWrite(e, 0);  
    digitalWrite(f, 1);  
    digitalWrite(g, 1);  
    delay(100);  
}  
void cinco() {  
    digitalWrite(a, 1);  
    digitalWrite(b, 0);  
    digitalWrite(c, 1);  
    digitalWrite(d, 1);  
    digitalWrite(e, 0);  
    digitalWrite(f, 1);  
    digitalWrite(g, 1);  
    delay(100);  
}  
void seis() {  
    digitalWrite(a, 1);  
    digitalWrite(b, 0);  
    digitalWrite(c, 1);  
    digitalWrite(d, 1);  
    digitalWrite(e, 1);  
    digitalWrite(f, 1);  
    digitalWrite(g, 1);  
    delay(100);  
}  
void sete() {  
    digitalWrite(a, 1);  
    digitalWrite(b, 1);  
    digitalWrite(c, 1);  
    digitalWrite(d, 0);  
    digitalWrite(e, 0);  
    digitalWrite(f, 0);  
    digitalWrite(g, 0);  
    delay(100);  
}  
void oito() {  
    digitalWrite(a, 1);  
    digitalWrite(b, 1);  
    digitalWrite(c, 1);
```

```

digitalWrite(d, 1);
digitalWrite(e, 1);
digitalWrite(f, 1);
digitalWrite(g, 1);
delay(100);
}
void nove() {
  digitalWrite(a, 1);
  digitalWrite(b, 1);
  digitalWrite(c, 1);
  digitalWrite(d, 1);
  digitalWrite(e, 0);
  digitalWrite(f, 1);
  digitalWrite(g, 1);
  delay(100);
}

void inicio() {
  for (int y = 0; y < 2; y++) { //Repetir duas vezes
    tone(buzzerPin, 330, 200); //Frequência e duração da nota Mi
    digitalWrite(ledRed, HIGH); //Liga o LED vermelho
    delay(100); //Intervalo de 200 milissegundos
    tone(buzzerPin, 330, 200); //Frequência e duração da nota Mi
    digitalWrite(ledYel, HIGH); //Liga o LED amarelo
    delay(100); //Intervalo de 200 milissegundos
    tone(buzzerPin, 330, 200); //Frequência e duração da nota Mi
    digitalWrite(ledGre, HIGH); //Liga o LED verde
    delay(100); //Intervalo de 300 milissegundos
    tone(buzzerPin, 262, 300); //Frequência e duração da nota Dó
    delay(100); //Intervalo de 200 milissegundos
    digitalWrite(ledRed, LOW); //Desliga o LED vermelho
    digitalWrite(ledYel, LOW); //Desliga o LED amarelo
    digitalWrite(ledGre, LOW); //Desliga o LED verde
    delay(500); //Intervalo de 0,5 segundos
  }
}

void perdeu() {
  for (int x = 0; x < 3; x++) { //Repetir 3 vezes
    for (int z = 0; z < 3; z++) { //Repetir 2 vezes
      tone(buzzerPin, 262, 200); //Frequência e duração da nota Dó
      digitalWrite(ledRed, HIGH); //Liga LED vermelho
      digitalWrite(ledYel, HIGH); //Liga LED amarelo
      digitalWrite(ledGre, HIGH); //Liga LED verde
    }
  }
}

```

```
    delay(250); //Intervalo de 250 milissegundos
    digitalWrite(ledRed, LOW); //Desliga o LED vermelho
    digitalWrite(ledYel, LOW); //Desliga o LED amarelo
    digitalWrite(ledGre, LOW); //Desliga o LED verde
}
tone(buzzerPin, 528, 300); //Frequência e duração da nota Dó
delay(100); //Intervalo de 100 milissegundos
}
}
```

```
void ganhou() {
    for (int w = 0; w < 2; w++) { //Repetir 2 vezes
        tone(buzzerPin, 440, 200); //Frequência e duração da nota Lá
        digitalWrite(ledRed, HIGH); //Liga led vermelho
        digitalWrite(ledYel, HIGH); //Liga led amarelo
        digitalWrite(ledGre, HIGH); //Liga led verde
        delay(250); //Intervalo de 250 milissegundos
        digitalWrite(ledRed, LOW); //Desliga led vermelho
        digitalWrite(ledYel, LOW); //Desliga led amarelo
        digitalWrite(ledGre, LOW); //Desliga led verde
        tone(buzzerPin, 495, 200); //Frequência e duração da nota Si
        digitalWrite(ledRed, HIGH); //Liga led vermelho
        digitalWrite(ledYel, HIGH); //Liga led amarelo
        digitalWrite(ledGre, HIGH); //Liga led verde
        delay(250); //Intervalo de 250 milissegundos
        digitalWrite(ledRed, LOW); //Desliga led vermelho
        digitalWrite(ledYel, LOW); //Desliga led amarelo
        digitalWrite(ledGre, LOW); //Desliga led verde
        tone(buzzerPin, 528, 200); //Frequência e duração da nota Dó
        digitalWrite(ledRed, HIGH); //Liga led vermelho
        digitalWrite(ledYel, HIGH); //Liga led amarelo
        digitalWrite(ledGre, HIGH); //Liga led verde
        delay(250); //Intervalo de 250 milissegundos
        digitalWrite(ledRed, LOW); //Desliga led vermelho
        digitalWrite(ledYel, LOW); //Desliga led amarelo
        digitalWrite(ledGre, LOW); //Desliga led verde
    }
}
```

```
    delay(500); //Intervalo de 500 milissegundos  
  }  
}
```

Para melhor compreensão do código acompanhe os seguintes passos:

#### 1- Declarar as variáveis

Assim como nos projetos anteriores, iniciamos a programação declarando as variáveis. Utilizamos as variáveis `butRed`, `butYel` e `butGre` para representar os botões vermelho, amarelo e verde, e atribuímos a elas os pinos analógicos A5, A4 e A3, respectivamente.

Em seguida, declaramos as variáveis `ledRed`, `ledYel` e `ledGre` atribuindo as portas digitais em que os LEDs se encontram conectados, ou seja, as portas 13, 12 e 11, respectivamente.

Em seguida, declaramos uma variável para cada segmento do display. Conforme o esquemático elétrico, atribuiremos a porta 4 a variável `a`; a porta 5 a variável `b`; a porta 6 a variável `c`; a porta 7 a variável `d`; a porta 8 a variável `e`; a porta 9 a variável `f`; a porta 10 a variável `g`.

Logo após, atribuímos a porta 3, em que o buzzer está conectado, a variável `buzzerPin`. Para que o buzzer reproduza sons diferentes para cada cor de LED declaramos as variáveis `tomRed`, `tomYel`, e `tomGre` atribuindo a elas o valor da frequência da nota musical desejada. Observe que para declarar estas variáveis utilizamos a diretiva `#define`, que permite dar um nome a um valor constante antes de o programa ser compilado. Nesse caso, as constantes definidas no Arduino não ocupam espaço na memória de programa do chip. Para saber mais sobre essa diretiva acesse:



<https://www.arduino.cc/reference/pt/language/structure/further-syntax/define/>.

Declaramos também a variável `rodada` para armazenar o número de rodadas do jogo, `led` para guardar o valor randômico que representará qual LED deve ser acionado, `temp_led_ligado` para armazenar o tempo em que o LED deve permanecer ligado (600 milissegundos), e `temp_led_desligado` para armazenar o tempo em que o LED deve permanecer desligado (100 milissegundos).

Além destas variáveis, declaramos também o vetor `mat` com 10 posições para armazenar as sequências do jogo e a variável `posi_mat` para armazenar a posição do vetor.

## 2- Configurar os pinos de entrada e saída e inicializar a comunicação serial

As variáveis `butRed`, `butYel` e `butGre` foram configuradas como entrada ( **INPUT** ). Por sua vez, as variáveis `ledRed`, `ledYel`, `ledGre`, `a`, `b`, `c`, `d`, `e`, `f`, `g` e `buzzerPin` foram configuradas como saída ( **OUTPUT** ).

A comunicação serial foi inicializada por meio da instrução: **Serial.begin(9600);** .

## 3- Inicializar o gerador de números aleatórios e chamar função `inicio()`

O gerador de números aleatórios foi inicializado através da função **randomSeed(analogRead(A0))** que utiliza como entrada aleatória a leitura analógica do pino A0.

Ainda no `setup`, chamamos a função `inicio()` que tocará uma melodia e acionará os LEDs para representar o início do jogo.

## 4- Gerar um valor aleatório e armazenar no vetor

Através da instrução `led = random(0, 3);` geramos um valor aleatório (0, 1 ou 2) e armazenamos na variável `led` . O valor de `led`

será armazenado no vetor `mat` na posição `posi_mat`.

Por exemplo, ao iniciar o jogo `posi_mat` será igual a 0 (zero), supondo que o número aleatório gerado pela função `random` tenha sido 1, esse valor será salvo na variável `led`. Desta forma, o valor de `led` (1) será armazenado no vetor `mat` na posição 0.

#### 5- Verificar o valor de `posi_mat`

Toda vez que o jogador acertar a sequência de cores e sons a variável `posi_mat` será incrementada em 1. Utilizamos o valor armazenado nessa variável como parâmetro da estrutura de controle de fluxo de seleção `switch...case`, comparando esse valor aos números especificados nos `cases`. Quando o valor armazenado em `posi_mat` for igual ao caractere especificado no `case`, a função para exibição do número correspondente no display deve ser executada.

#### 6- Criar estrutura que mostre a sequência dos LEDs sempre que passar o nível

A cada nível acertado pelo jogador, o jogo deverá repetir a sequência anterior e incluir mais um comando de LED aceso para ser memorizado. Por exemplo, no nível 0 apenas o LED verde foi acionado, no nível 1 o LED verde deve ser acionado novamente e mais um LED deverá ser acionado (verde, amarelo ou vermelho, dependendo do valor aleatório gerado), e assim sucessivamente.

Desta forma, criamos uma estrutura de repetição utilizando o `for` para percorrer o vetor de forma que mostre toda a sequência gerada.

#### 7- Criar estrutura que selecione os LEDs na sequência gerada

Para selecionar os LEDs na sequência gerada utilizamos a estrutura `switch... case`, que verificará o valor armazenado no vetor `mat` na posição `j`. Caso o conteúdo de `mat` na posição `j` seja igual a 0 o LED verde e seu tom serão acionados, caso `j` seja igual a 1 o

LED amarelo e seu tom serão acionados, e sendo `j` igual a 2 o LED vermelho e seu tom serão acionados.

- 8- Declarar variável temporária para aguardar o jogador digitar a sequência fornecida

Criamos a variável do tipo `int` para aguardar o jogador digitar a sequência fornecida, através da instrução: `int temp = 0;`

- 9- Criar estrutura de controle para aguardar o usuário pressionar o botão com a sequência

No momento em que o usuário pressiona o botão será feita a comparação se a sequência pressionada foi igual a gerada pelo jogo. Isso é feito através das estruturas *if* de verificação que estão dentro do *while*.

A cada acerto da sequência a variável `temp` é incrementada com o objetivo de efetuar a próxima comparação, se houver. Caso o jogador pressione a sequência incorreta será impresso uma mensagem de erro no monitor serial, a variável `posi_mat` será igualada a 11 e um `break` será executado para parar o laço de repetição *while*.

Para identificar se o jogador ganhou verificamos o valor da variável `posi_mat`. Se `posi_mat` for igual a 10 indica a vitória do jogador e a função `ganhou()` será chamada, tocando uma melodia.

Se o valor de `posi_mat` for maior ou igual a 11 indica que o usuário errou a sequência e a função `perdeu()` será chamada. Além disso, será exibida na serial a mensagem: " ---- >>> FIM DE JOGO <<< ----" .

Em ambas as situações, o jogo será reiniciado atribuindo zero a variável `posi_mat`.

- 10- Criar as funções dos números a serem exibidos no display 7 segmentos

Em seguida, criamos as funções que serão responsáveis por acionar os segmentos do display para que ele exiba números de 0 a 9. O display será utilizado para mostrar ao jogador o nível em que ele está, sendo assim, a cada sequência acertada o display incrementará em 1 o valor mostrado.

- 11- Criar as melodias nas funções `inicio()` , `perdeu()` e `ganhou()`.



### **OBSERVAÇÃO:**

- Utilize o cabo para bateria 9 V com plug P4 para alimentar sua placa UNO. Desse modo, você poderá jogar o Jogo da Memória sem a necessidade do computador e poderá leva-lo a qualquer lugar.

## **TINKERCAD**

Para melhor visualizar e/ou simular o circuito e a programação deste projeto basta se acessar o seguinte link:  
<http://www.blogdarobotica.com/tinkercad-JogodaMemoria>.

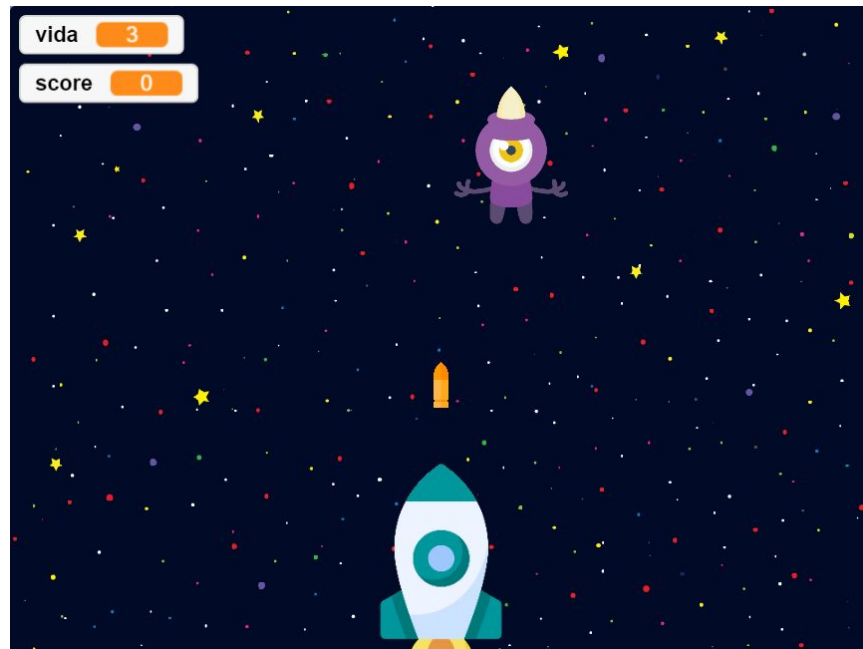
# BATALHA ESPACIAL

Você já pensou em desenvolver seu próprio jogo de vídeo game? Se a resposta foi sim, prepare-se, pois, este projeto foi feito para você. Neste projeto, você aprenderá a desenvolver um jogo de Batalha Espacial, usando o Pictoblox, um software de programação gráfica de fácil codificação e interação, em conjunto com a placa UNO. O interessante aqui é introduzir a possibilidade de unir o hardware com o software, o que torna o aprendizado muito mais prazeroso. Divirta-se!

A proposta deste projeto é desenvolver o jogo Batalha Espacial (Figura 25) utilizando a placa UNO e outros componentes eletrônicos como controle. A parte gráfica deste jogo será desenvolvida utilizando programação em bloco no PictoBlox, que se encontra disponível para download no seguinte link:

[Download PictoBlox](#)

**Figura 25** - Jogo Batalha Espacial.



## MATERIAIS NECESÁRIOS

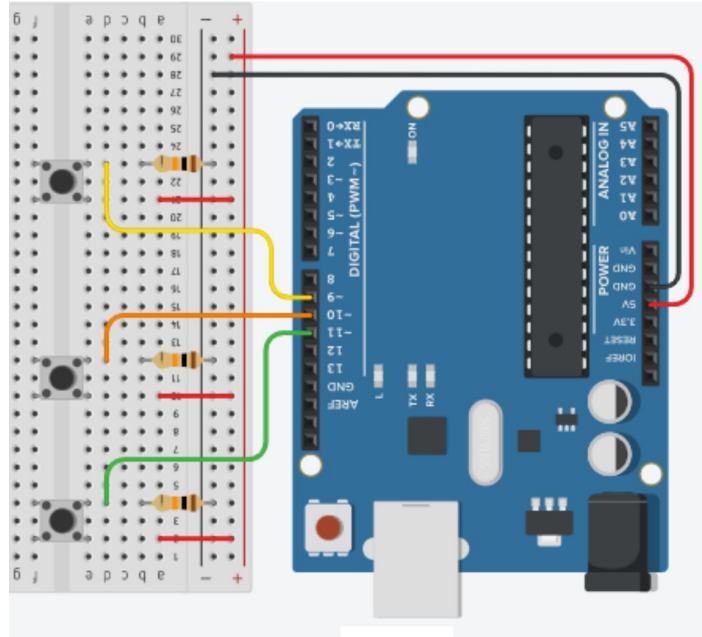
- 1 x [Placa UNO SMD R3 Atmega328 compatível com Arduino UNO](#);
- 1 x [Cabo USB](#);
- 1 x [Protoboard](#);
- 3 x [Resistores de 10 kΩ](#);
- 3 x [Botões do tipo push button](#);

## ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

Conforme dito anteriormente, vamos desenvolver o controle do jogo Batalha Espacial utilizando a placa UNO e outros componentes eletrônicos. Desta forma, conecte os componentes na protoboard,

como ilustra a Figura 26. Lembre-se que a placa UNO deve estar desconectada enquanto você monta o circuito.

**Figura 26** – Circuito para controle do jogo Batalha Espacial.



Ao montar seu circuito na protoboard preste atenção nos seguintes pontos:

- O primeiro botão encontra-se conectado ao pino digital 9 da placa UNO. Esse botão será responsável por movimentar o Foguete para a esquerda;
- O segundo botão encontra-se conectado ao pino digital 10 da placa UNO. Esse botão será responsável por movimentar o Foguete para a direita;
- O último botão está conectado ao pino digital 11 e será responsável por disparar as Balas do Foguete.

TINKERCAD

Para melhor visualizar o circuito deste projeto basta se acessar o seguinte link: <http://blogdarobotica.com/tinkercad-BatalhaEspacial>.

## CONSTRUÇÃO GRÁFICA

A parte gráfica do jogo Batalha Espacial vamos utilizar o PictoBlox. Para facilitar a construção gráfica dividimos a explicação em etapas.

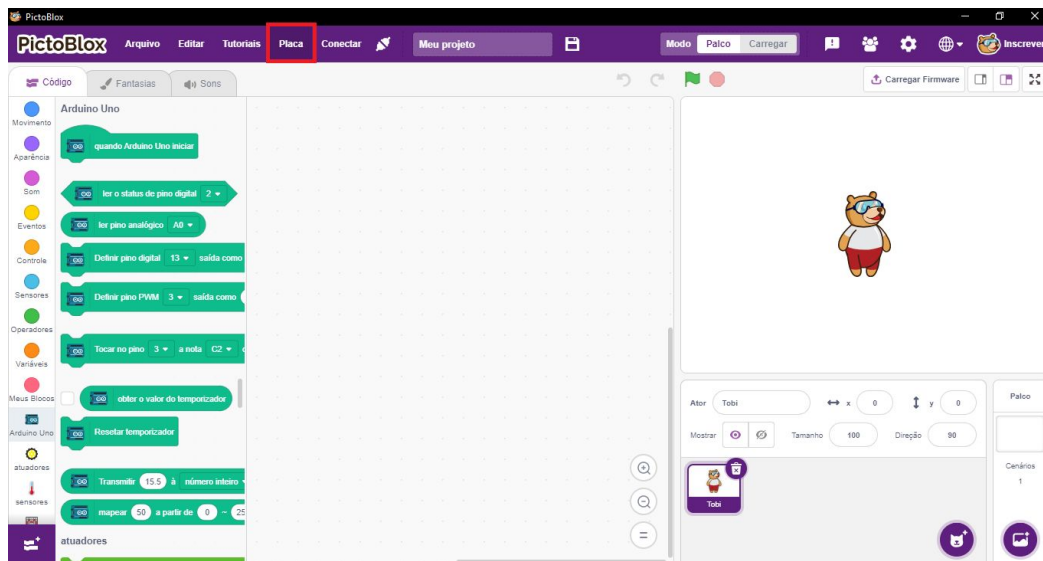
### Etapas 1: Seleção da placa e conexão

A primeira etapa para criação da Batalha Espacial no PictoBlox é a seleção da placa microcontroladora. Para isso, clique em Placa (Figura 27) na barra superior do programa. Em seguida busque por Arduino UNO (Figura 28).

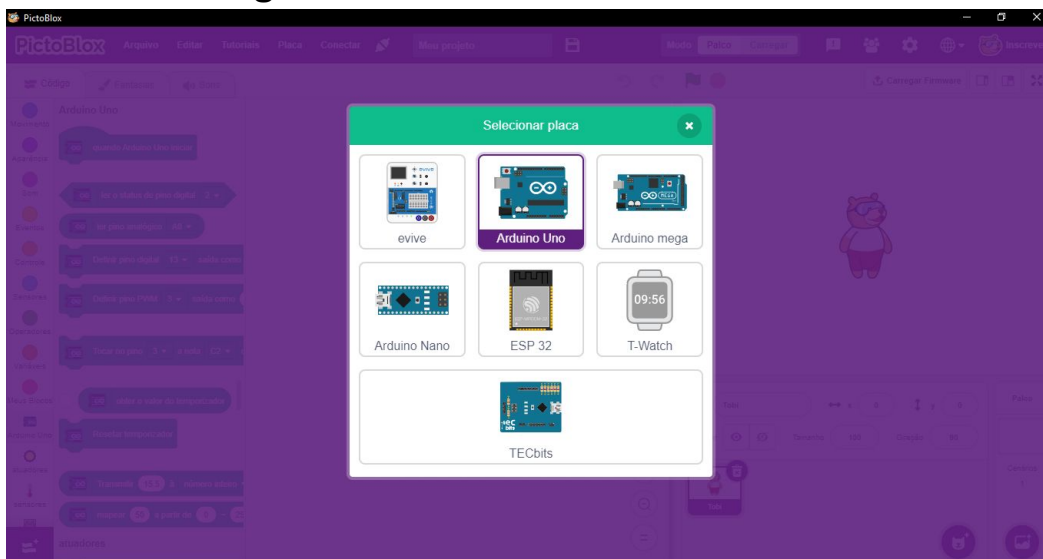
Para conectar a placa UNO ao programa, clique em Conectar na barra superior do PictoBlox. Logo após, selecione a porta serial em que a placa encontra-se conectada (Figura 29).

**Figura 27** - Seleção da placa no PictoBlox.

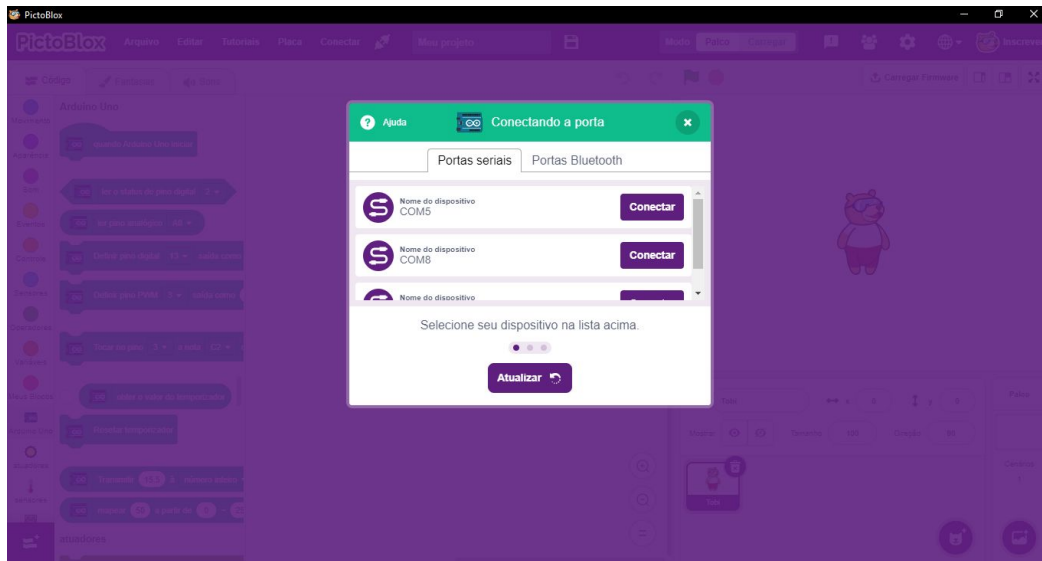




**Figura 28 - Selecione a Placa UNO.**



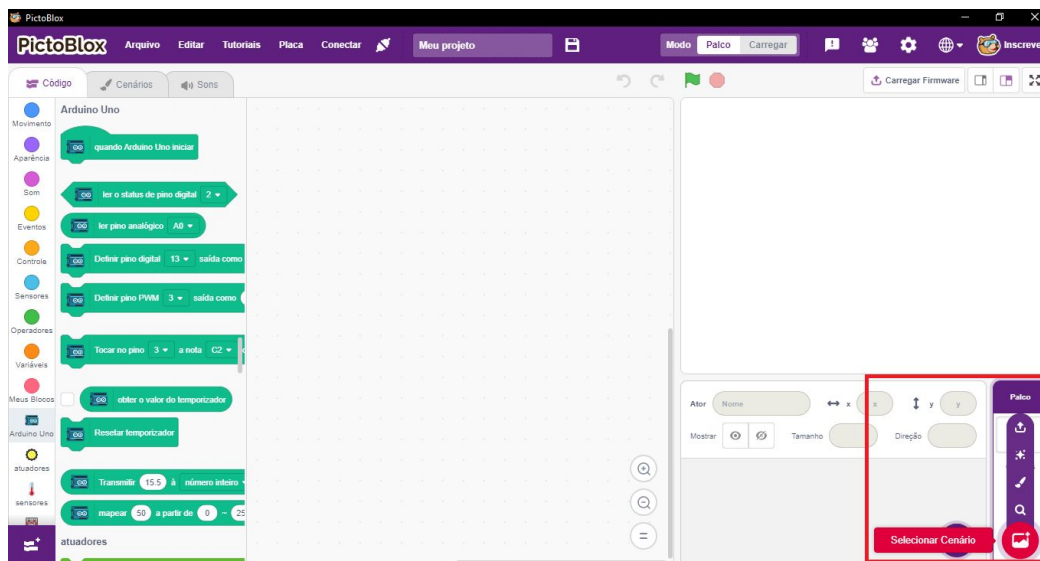
**Figura 29- Seleção da porta de comunicação.**



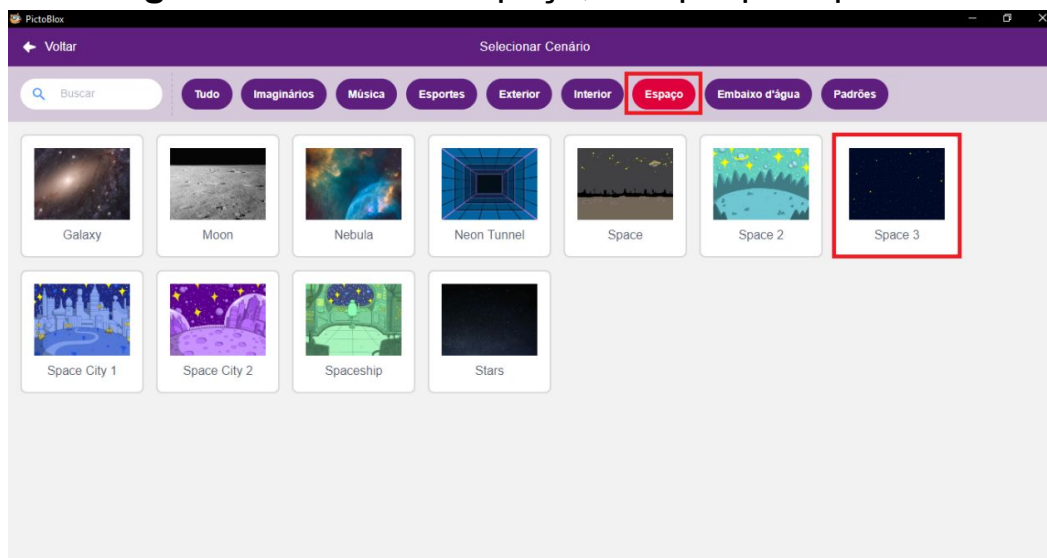
## Etapa 2: Configurando o cenário

Como a batalha ocorrerá no espaço, precisamos inserir um cenário que se pareça com o espaço sideral. Desta forma, clique em Selecionar Cenário (Figura 30) e busque na aba Espaço por Space3.

**Figura 30** – Selecionar cenário.



**Figura 31** - Na aba Espaço, busque por Space 3.

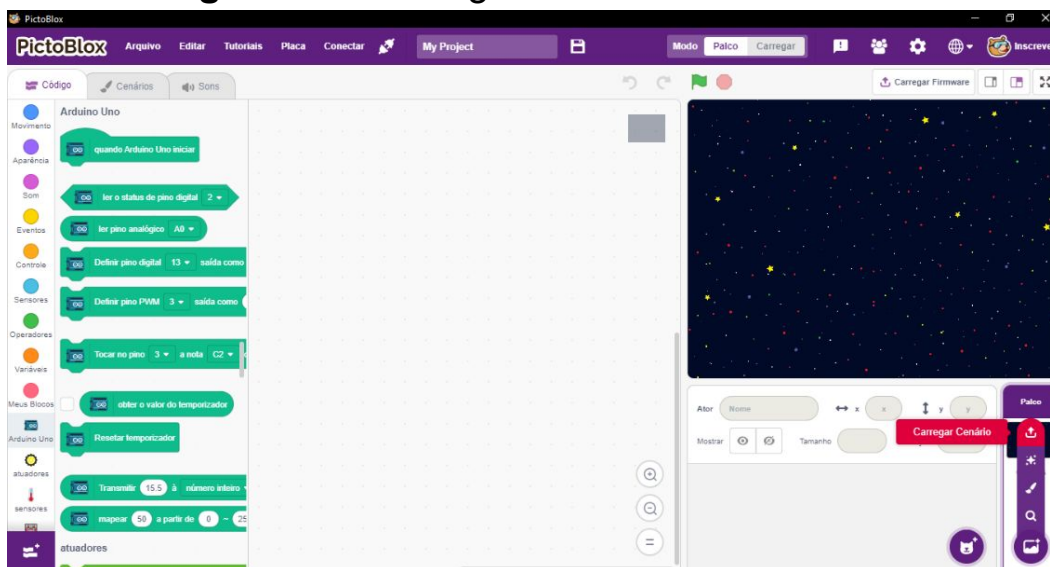


Para dar um ar profissional ao nosso game, vamos incluir também dois outros cenários, um de Apresentação – Batalha Espacial, e outro de Game Over. Estes cenários podem ser baixados através do seguinte link:

<http://www.blogdarobotica.com/Cenario-BatalhaEspacial>

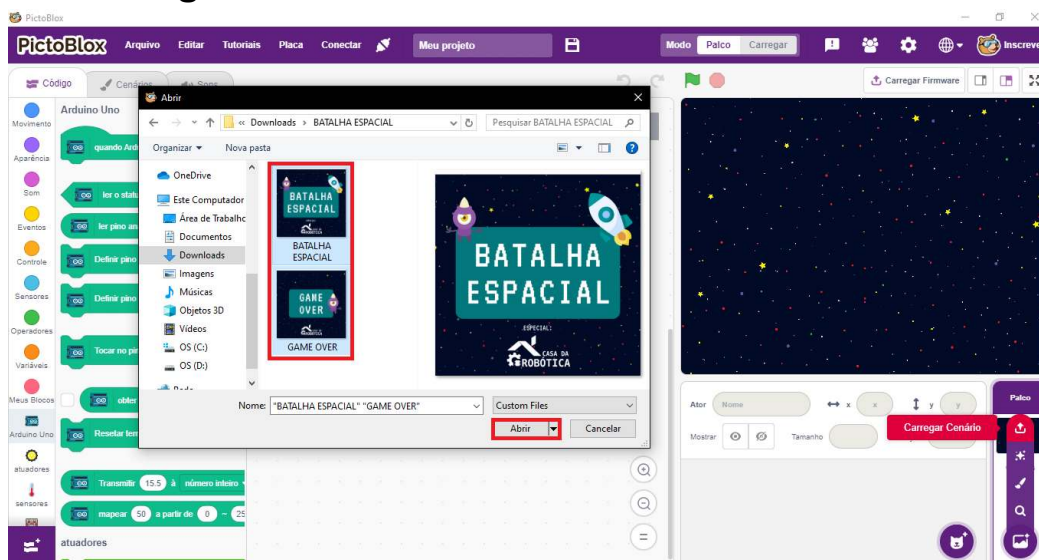
Para anexar os cenários de Apresentação e Game Over selecione a opção Carregar Cenário (Figura 32).

**Figura 32 - Carregar Cenário no PictoBlox.**



Em seguida, anexe os arquivos dos cenários baixados para o PictoBlox, conforme a Figura 33.

**Figura 33 - Anexando cenários no PictoBlox.**



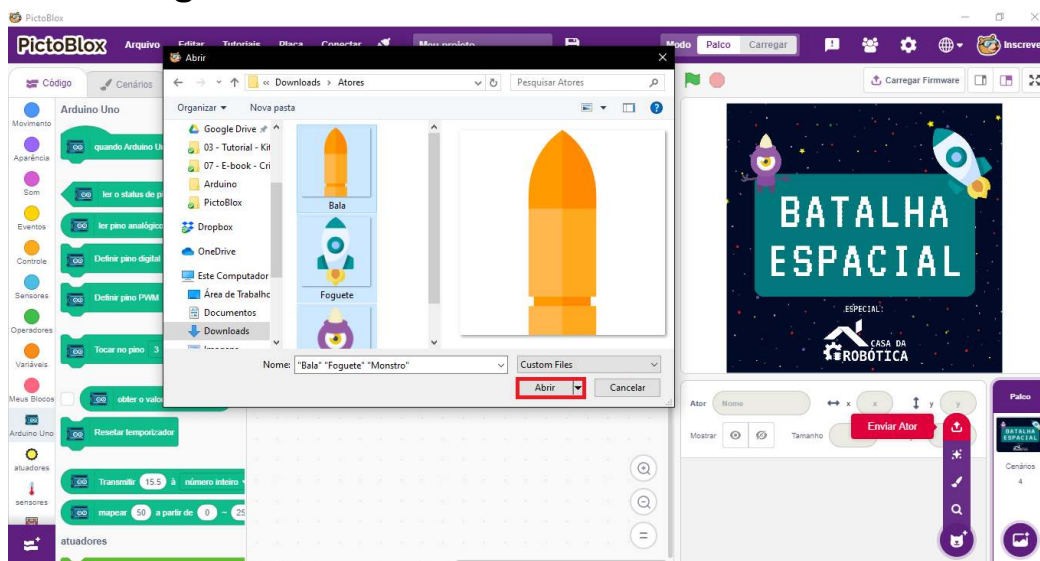
### Etapa 3: Seleção dos atores

Para este jogo, precisamos de 3 atores: Nossa nave espacial, nosso inimigo – o monstro espacial, e algo para destruí-lo – as balas espaciais. Você pode selecionar os atores na biblioteca do PictoBlox ou baixar os nossos através do seguinte link:

<http://www.blogdarobotica.com/Atores-BatalhaEspacial>

Para incluir os nossos atores clique na opção Carregar Ator e em seguida anexe os atores baixados, conforme a Figura 34.

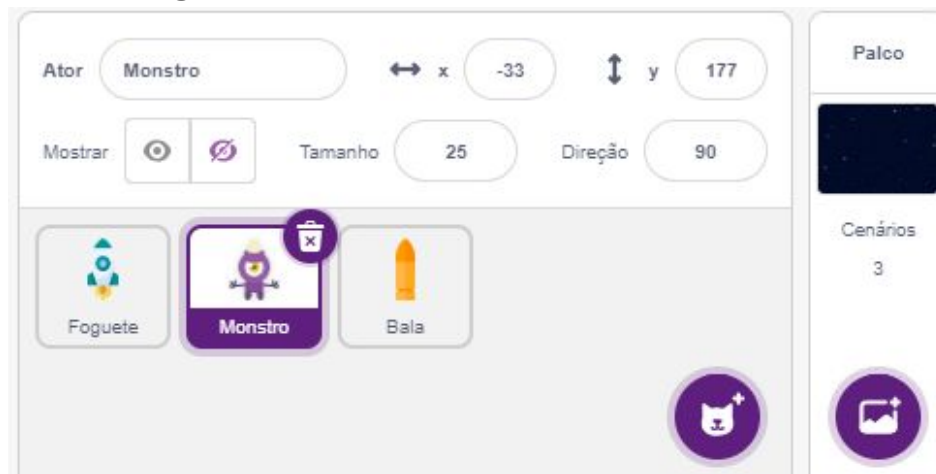
**Figura 34 - Anexando os atores no Pictoblox.**



Em seguida, na área de projeto selecione cada um dos autores para alterar o tamanho deles (Figura 35), atribuindo:

- Tamanho 50 para o Foguete;
- Tamanho 25 para o Monstro;
- Tamanho 10 para a Bala.

**Figura 35 - Alterar tamanho dos atores.**

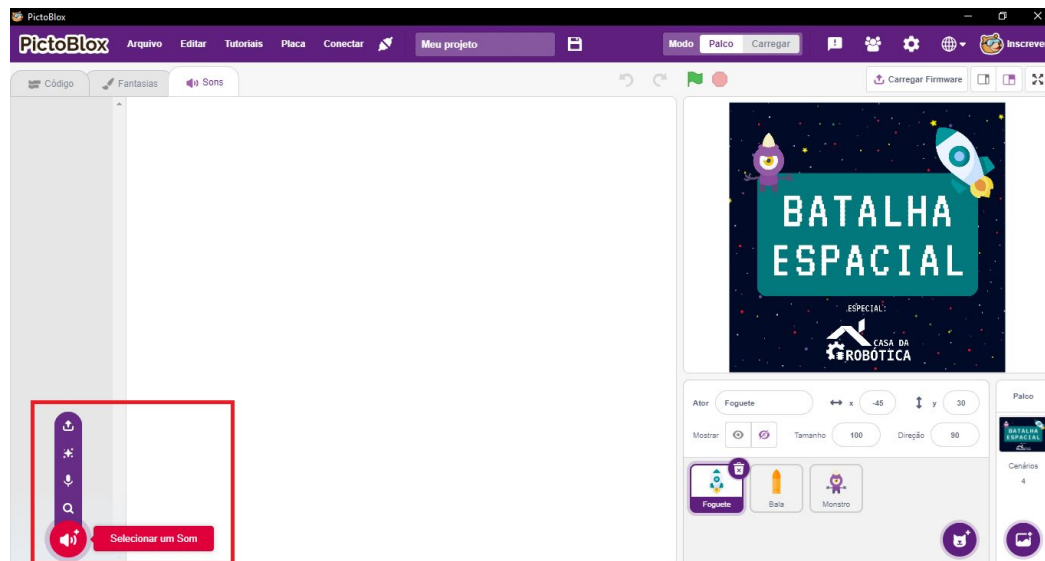


#### Etapa 4: Programação do Foguete

O PictoBlox é programado via blocos, o que o torna ideal para iniciantes no mundo maker. Desta forma, para realizar a programação precisamos arrastar os blocos para a área de script de código e encaixá-los.

Iniciamos o jogo Batalha Espacial programando as funções do Foguete, para isso clique sobre esse ator para selecioná-lo. O primeiro bloco a ser arrastado é o “Quando o sinalizador for tocado”. Em seguida, incluímos a música de fundo do jogo. Para isso, clique na aba Som, clique em Selecionar um Som (Figura 36) e busque por Vídeo Game 2 na biblioteca.

**Figura 36 - Selecionar um som no PictoBlox.**



Logo após, usamos o bloco Esconder para que os atores não apareçam na tela de Apresentação – Batalha Espacial. Feito isso, incluímos o cenário de Apresentação – Batalha Espacial, aguardamos 3 segundos e mudamos o cenário para Space 3. Em seguida, incluímos o bloco Mostrar para que os atores apareçam na tela novamente e posicionamos o Foguete para a posição ( $x = 0$ ,  $y = 140$ ).

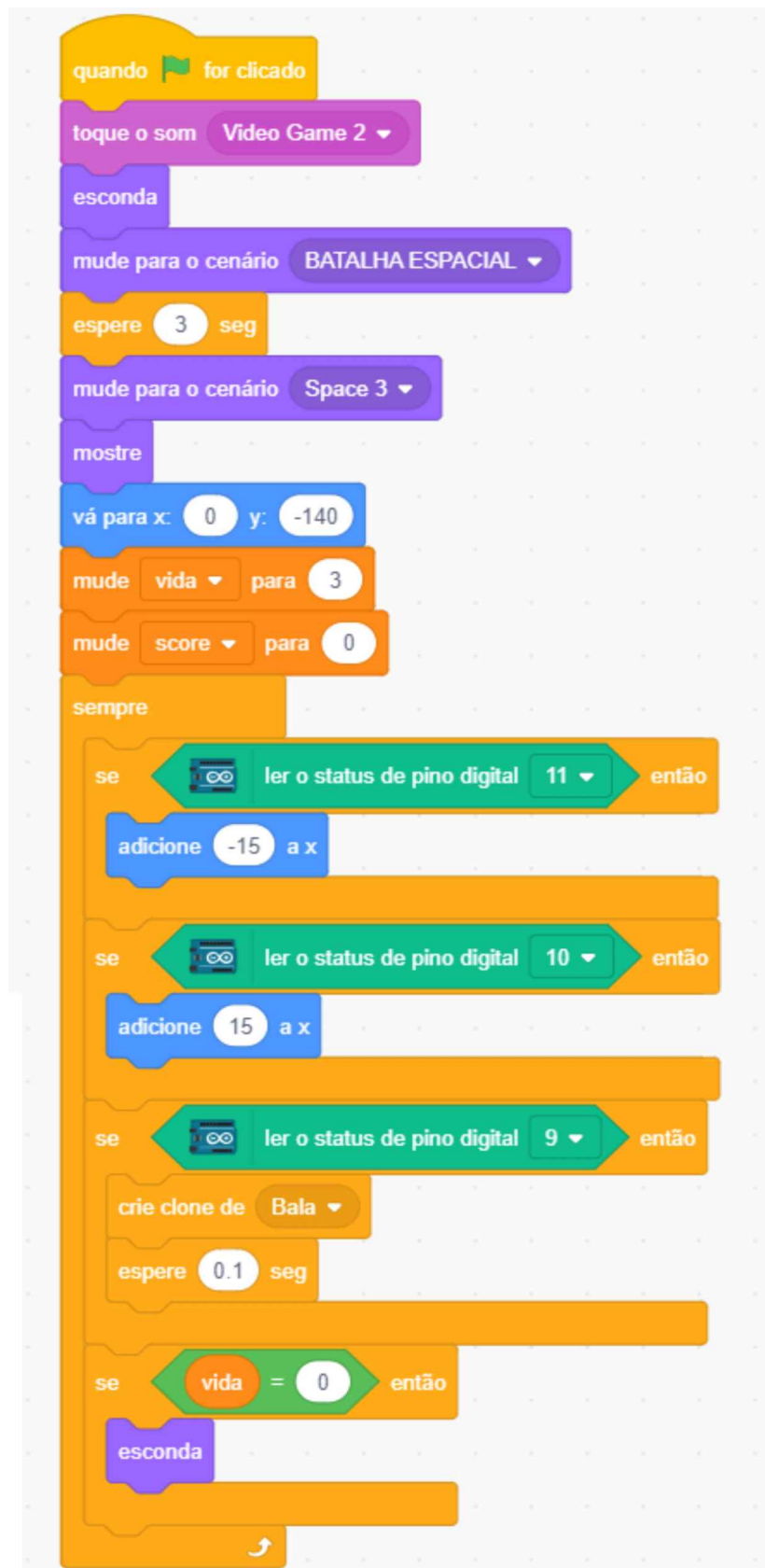
Na aba Variáveis criaremos as variáveis vida e score. Usaremos vida para armazenar as chances que o jogador tem, de modo que se o monstro alcançar o Foguete uma vida será perdida. Por sua vez, a variável score marcará a quantidade de Monstros atingidos.

Em seguida, incluímos as leituras dos botões conectados a placa UNO para efetuar a movimentação do foguete e disparo das balas.

A Figura 37 ilustra a programação necessária para o Foguete.

**Figura 37** - Programação para o Foguete.







## Etapa 5: Programação do monstro

Escreveremos dois scripts de programação do monstro: Um para criar vários monstros e outro para atribuir ações a eles, conforme a Figura 38.

**Figura 38** - Programação para o Monstro.





Etapa 6: Programando o disparo das balas

Quando o botão de disparo for clicado um clone de Bala será criado. Desta forma, iniciamos o script de Bala incluindo o bloco Quando eu começar como um clone. Em seguida, incluímos o toque Laser2, que deve ser tocado sempre que houver o disparo. Logo após, introduzimos a posição inicial da Bala, que deve ser a mesma que a do Foguete, para parecer que o mesmo está disparando-a. Por fim, inserimos os blocos responsáveis pela movimentação da bala.

A Figura 39 demonstra a programação final para a Bala.

**Figura 39** - Programação para a Bala.



## Etapa 7: Enviar firmware para a placa UNO

Para fazer o upload do código na placa UNO clique em Carregar Firmware (), localizado no canto superior direito do PictoBlox.

**Figura 40** - Carregar Firmware no PictoBlox.



Para melhor visualizar e/ou simular a Batalha Espacial basta acessar o seguinte link:

<http://blogdarobotica.com/jogo-BatalhaEspacial>.



# DESAFIOS

## DESAFIO 1

O que acha de adicionar um buzzer no game STOP – Um Jogo com LEDS? Você pode criar um efeito sonoro para quando o jogador passar de fase ou perder o jogo.

## DESAFIO 2

Também achou que o Jogo da Memória está muito fácil? Vamos dificultar um pouco? Propomos como desafio que você implemente mais um botão e um LED neste jogo, assim o jogador terá mais uma cor para memorizar.

## DESAFIO 3

Na Batalha Espacial, que tal adicionar um novo monstro? Assim, o jogo ficará mais dinâmico. Outro desafio é criar um sistema de vidas, que possa surgir de tempos em tempos para repor ou aumentar o número de vidas do Foguete.

## DESAFIO 4

Que tal criar um jogo diferente no PictoBlox agora? Pode ser no estilo do Flipboard Game ou até mesmo o jogo do dinossauro do navegador Chrome... As possibilidades aqui são infinitas! Podemos usar botões para pular ou até mesmo sensores, ex. sensores

ultrassônicos, piezo, células de carga, entre outros. Use a criatividade, afinal “a criatividade é a inteligência se divertindo” (Albert Einstein).

Se conseguir implementar um desses desafios, saiba que queremos ver! Faça um vídeo e nos marque no instagram: @casadarobotica !

Boa sorte!





# CONSIDERAÇÕES FINAIS

Chegamos ao final do nosso e-book **Criando Jogos com Arduino – Passo a Passo**.

Esperamos que este material tenha auxiliado e contribuído no seu aprendizado de eletrônica e programação.

Caso tenha alguma dúvida entre em contato conosco, afinal “a dúvida é o princípio da sabedoria” (Aristóteles).

Caso você tenha alguma sugestão, por favor, entre em contato conosco. Sua opinião é muito importante para nós.

[contato@casadarobotica.com](mailto:contato@casadarobotica.com)

Acompanhe as novidades em nossas redes sociais:

Facebook: @casadaroboticaoficial

Instagram: @casadarobotica

Grupos para compartilhamento de informações, dúvidas e novidades:

Telegram: <http://www.blogdarobotica.com/GrupoTelegram>

Whatsapp: <http://www.blogdarobotica.com/GrupoWhatsapp>

Até a próxima,

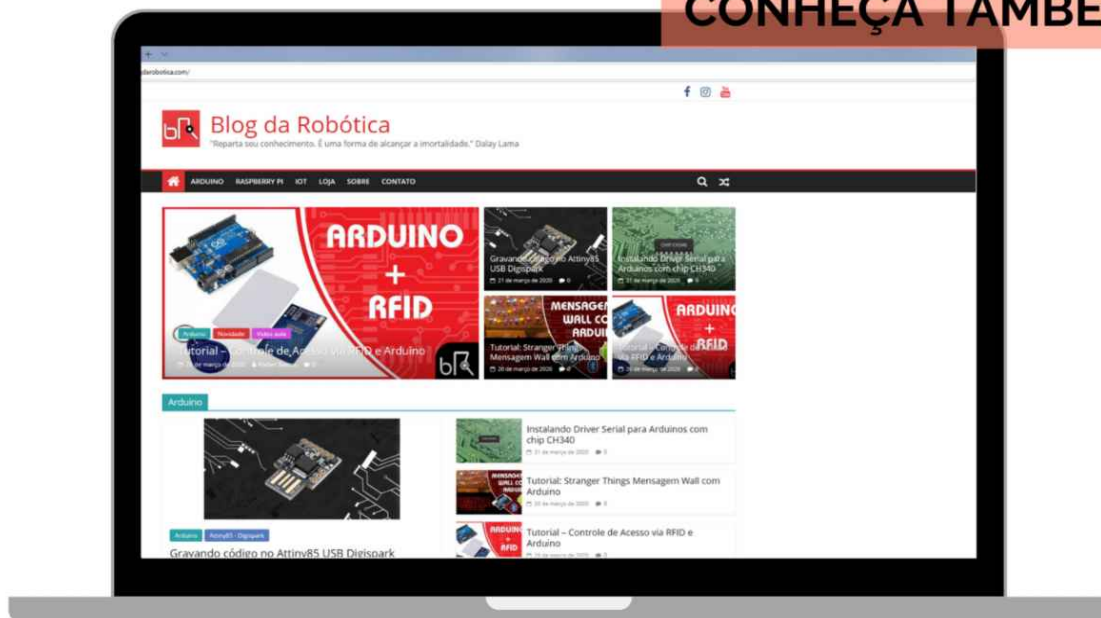
## **Equipe Casa da Robótica**



# blog da ROBOTICA

#ELETRÔNICA #MAKER #Tecnologia #PROGRAMAÇÃO  
#CIÊNCIAS #ROBÓTICA #INOVAÇÃO

CONHEÇA TAMBÉM



blogda**robotica**.com



REFERÊNCIAS

# REFERENCIAS

**ARDUINO.** Disponível em: <https://www.arduino.cc/>

MONK, Simon. **Programação com Arduino:**  
Começando com Sketches. 2. ed. Bookman, 2017.  
180 p.

**PICTOBLOX.**

<https://thetempedia.com/product/pictoblox/download-pictoblox/>



# blog da ROBOTICA

#ELETRÔNICA #MAKER #Tecnologia #PROGRAMAÇÃO  
#CIÊNCIAS #ROBÓTICA #INOVAÇÃO

CONHEÇA TAMBÉM



blogda**robotica**.com











---

[WWW.CASADAROBOTICA.COM](http://WWW.CASADAROBOTICA.COM)  
[CONTATO@CASADAROBOTICA.COM](mailto:CONTATO@CASADAROBOTICA.COM)

(77) 99151-2820

FACEBOOK: @CASADAROBOTICAOFICIAL

INSTAGRAM: @CASADAROBOTICA

---

FINAL DO JOGO





